# "SPRITES" Y GRAFICOS EN LENGUAJE MAQUINA

(ZX Spectrum)

Librería LARA, S, A.
Fuente Dorada, 17
47001 Valladolid

# “Sprites” y gráficos en lenguaje máquina

(ZX Spectrum)

**John Durst**



ANAYA MULTIMEDIA

MICROINFORMATICA

Título de la obra original:
MACHINE CODE SPRITES AND GRAPHICS FOR THE ZX SPECTRUM

Traducción de: Pedro Soria
Diseño de colección: Antonio Lax
Diseño de la cubierta: Narcís Fernández



First Published in English 1984 by:
Sunshine Books (an imprint of Scot Press Ltd)
12/13 Little Newport Street
London WC2R 3LD


Villafranca, 22. 28028 Madrid
Depósito Legal: M. 16.714-1985
I.S.B.N.: 84-7614-017-7
Printed in Spain
Imprime: Gráf. FUTURA, Soc. Coop. Ltda.
Villafranca del Bierzo, 21-23. Fuenlabrada (Madrid)

# Indice

# Contenido en detalle

## 6. Caracteres pequeños

Caracteres de seis bits.—Caracteres de cuatro bits.—Máquina de escribir.—Operaciones lógicas.

## 7. Generación de caracteres de seis bits

Diseño de un contador.—Desplazamiento de las letras.—Almacenamiento, en memoria, de los bytes.—Utilización de los caracteres de seis bits.

## 8. Animación. Figuras móviles

Realización de gráficos móviles.

## 9. El movimiento del "sprite"

Organización del fichero de imagen.—Realización de un carácter desplazado verticalmente.—Realización de un carácter desplazado horizontalmente.—Realización de un "sprite".

## 10. Realización de los fondos sobre los que se mueven los "sprites"

Protección del dibujo de fondo de la pantalla.—"Sprites" con capacidad para moverse libremente.—Impresión de "sprites" sobre un fondo.

## 11. El fichero de atributos

El color.

## 12. El fichero de imagen

Búsqueda de los UDG's.—Deslizamiento del contenido de la pantalla ("scrolling").—Reorganización del fichero.—Reducción de la imagen.

## 13. Entradas y salidas

Modos de interrupción.—La pantalla de televisión.—El generador de tonos del Spectrum.

# Introducción

Este libro trata de las imágenes que puedes componer con el Spectrum. Se explica cómo puedes realizar modelos de imágenes y letras, de una complejidad y variedad de colores, que no podrías conseguir usando la programación BASIC normal.

Podrás aprender a dibujar tus propios *sprites*[1] y cómo moverlos por la pantalla —totalmente independientes del fondo de la imagen—. Aprenderás a realizar letras con unas formas que están fuera de tu alcance con los caracteres normalmente realizables por el Spectrum —letras grandes (de tamaño desde dos a ocho veces el tamaño de las letras normales), letras pequeñas (que permiten la colocación de 40 o incluso 64 caracteres por línea) y letras inclinadas—. Todas estas aplicaciones las podrás usar en tus programas, para con ello hacerlos más interesantes y divertidos.

Para poder realizar todo esto, tendrás que profundizar en el funcionamiento interno de tu Spectrum para saber cómo se organizan el conjunto de caracteres, los ficheros de pantalla y los de atributos. Al finalizar este libro, serás capaz de manejar todas estas cosas de una manera tan espectacular y limpia como la forma en que un prestidigitador maneja sus platos y botellas. Una vez que comprendas cómo hacerlo, así como todas las cosas que puedes hacer con estas nuevas posibilidades, serás capaz de usar tu Spectrum para realizar cosas que ni siquiera la Sinclair Research había soñado.

Las técnicas descritas en este libro usan el código máquina[2] casi todo el tiempo, con el que se evitan las restricciones inherentes al lenguaje BASIC.

---

[1] Se denomina con la palabra *sprites* a las figuras que representan un elemento de la imagen con una movilidad propia e independiente del resto de los elementos de la imagen.

[2] Código máquina es la denominación que se da a un lenguaje de diálogo con los ordenadores. Este lenguaje es de nivel más bajo que el BASIC y, por tanto, de menos potencia que éste, pero presenta ventajas al poder realizarse programas más rápidos que los realizados con el BASIC.

Aunque se supone, en este libro, que conoce algo de la programación en el lenguaje máquina, he intentado desarrollar las rutinas de una manera tan sencilla y lógica como me ha sido posible: espero que todos los lectores serán capaces de seguir el desarrollo de cada programa, paso a paso.

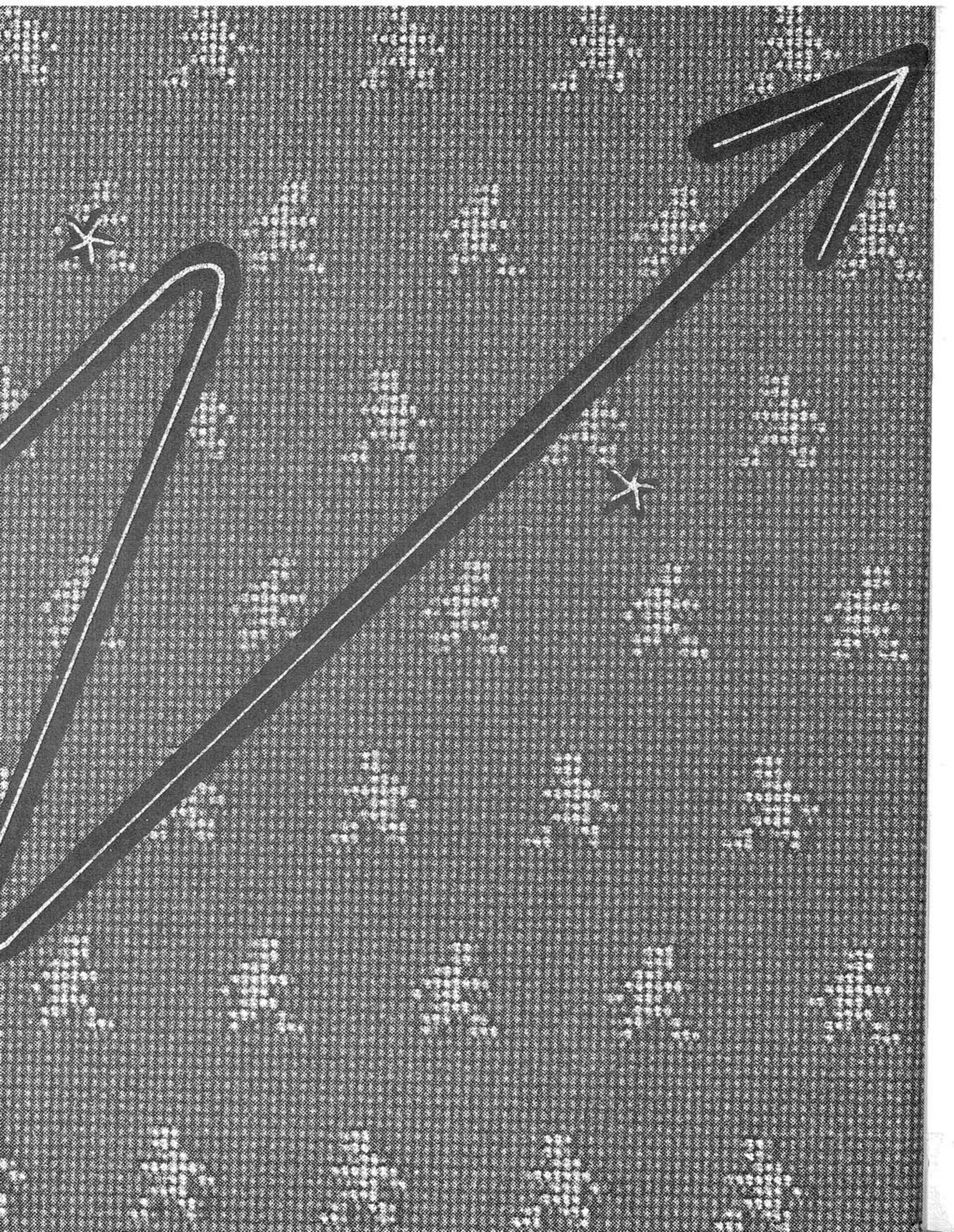
Casi todas las rutinas pueden funcionar solas: es decir, pueden ser programadas y ejecutadas individualmente para unos determinados resultados. Pero sólo unas pocas de ellas están pensadas para esta forma solitaria de funcionamiento. Están pensadas para que las uses —para adaptarlas e incorporarlas en tus programas—, y así ofrecerte nuevas posibilidades para realizar mejores programas. Las rutinas hacen lo que tú quieras que hagan.

Espero que este libro sea útil para ti y, sobre todo, que te divierta.

## Notas de programación

Es importante tener cuidado, al introducir los programas, teclear correctamente todos los caracteres. Sobre todo, el número 1 y la l minúscula, comas y puntos, pueden parecer similares.

Las referencias a páginas del *Manual del Spectrum* se refieren a la segunda edición (1983).



# 1
# Código máquina

Cualquier escritor ha de comenzar haciendo algunas suposiciones, y la mía es que los lectores de este libro no serán unos completos novatos en la programación en lenguaje máquina. Tú no pensarás que el «registro BC» es una lista de familias con unas raíces de 2.000 años de antigüedad, o que «desplazamiento lógico a la derecha» tiene algo que ver con la política.

Hay numerosos libros que tratan bien el tema de la iniciación al código máquina, los cuales te introducirán en las reglas del sistema y te explicarán cómo conseguir resultados interesantes. Si te sientes atraído (y probablemente deberás estarlo para haber comprado este libro), tendrás un par de esos libros, los cuales te explicarán el lenguaje máquina del Z80 en lo referente al Sinclair ZX Spectrum. Pero, además, me atrevería a decirte que hay al menos dos obras necesarias (o casi necesarias) que deberías de echar de menos en tus estanterías.

La primera de esas «necesidades» es *The complete Spectrum ROM Disassembly* (Melbourne House), escrito por los doctores Ian Logan y Frank O'Hara. La ROM (memoria de lectura, es decir, no se puede escribir en ella) es un cofre de tesoros en rutinas que pueden llamarse para ser usadas en sus programas (algunas de estas rutinas serán mencionadas en este libro). El gran valor de estas rutinas es que están en el Spectrum, ya depuradas, por lo que no necesitas saber cómo trabajan —sólo necesitas conocer lo que hacen y su dirección de comienzo—. Sin embargo, es previsible que a veces desees cambiar las normas de uso de estas rutinas, por ejemplo, entrando en una rutina en un punto posterior al normal de entrada.

Para poder comprender de una forma más profunda lo que realmente hacen estas rutinas y cómo funcionan, el mejor método es consultar sus listados originales, esto es lo que nos permite realizar el *Spectrum ROM Disassembly* (este paquete de *software* permite el paso del código máquina de la ROM, formado por ceros y unos, al ensamblador).

El otro libro necesario sería el de Rodnay Zaks, titulado *Programación del Z80* (Anaya Multimedia). Este es un gran libro que contiene *todo* lo que concierne al microprocesador del Spectrum; si quieres saber qué es lo que ocurre al *flag* de la paridad después de alguna operación un tanto extraña (o tal vez qué es el *flag*, o indicador, de paridad), entonces Zaks es tu hombre.

He intentado en este libro simplificar lo más posible los puntos de mayor dificultad, pero el código máquina es un pequeño laberinto y, por tanto, no deberás dudar en buscar otras explicaciones más detalladas, que en algunas ocasiones te aclararán más algún punto.

Casi todas las rutinas que se comentan y utilizan en este libro tienen como dirección de comienzo la F000h[1] —o algunas veces las F100h o F200h—. Estas direcciones son arbitrarias; en realidad, como sugerencia, las rutinas que forman parte de un programa mayor es conveniente ponerlas en las direcciones adecuadas para que formen en la memoria un bloque con el resto de su programa.

Los programas se han ejecutado en un Spectrum de 48K[2] de memoria. Sin embargo, la casi totalidad de ellos, excepto aquellos que usan una gran cantidad de memoria, trabajarán también en un Spectrum de 16K, pero, para evitar desordenar las páginas ya amontonadas, yo aconsejaría no listar las direcciones alternativas.

Si *estás* utilizando un Spectrum de 16K, como regla general se puede utilizar lo siguiente: donde el Spectrum de 48K tiene como dirección de comienzo la F000h, el Spectrum de 16K deberá tener como dirección de comienzo la 7000h.

Por tanto, para fijar las rutinas en el contexto de los 16K deberías de poner "7" donde se encuentre "F" como primer dígito de una dirección hexadecimal, pero será también necesario chequear todo el programa cuidadosamente, para asegurarte de que este cambio no te supone ningún problema posterior a la hora de su ejecución.

## Hexadecimal o decimal

El párrafo anterior introduce un nuevo asunto que es necesario que tratemos. Se trata de la gran controversia entre hexadecimal y decimal. En el libro he usado direcciones en hexadecimal porque parece una regla sencilla para seguir el código máquina, aunque, cuando programamos en BASIC, la notación decimal es realmente la única opción posible en el Spectrum. La entrada limitada en binario es usada principalmente para la creación de gráficos.

Surge la pregunta ahora: ¿Por qué no utilizar la numeración decimal igualmente en todas las situaciones?

La razón es que, en código máquina, la notación decimal da una impresión muy pobre del binario del que procede y que es el código que entiende la máquina.

Por ejemplo, los números decimales 19, 27, 35 y 43 son todos instrucciones del Z80. Estos números no nos dan una imagen de semejanza que es propia de una familia, aunque estos códigos pertenecen a las instrucciones "incrementar DE", "decrementar DE", "incrementar HL" y "decrementar HL". Pero expresados como números hexade-

[1] La h al final de los cuatro dígitos indica que esta numeración se da en código hexadecimal.
[2] Un K es una unidad de medida de memoria y es equivalente a 1.024 bytes.

cimales se pondrían como 13, 1B, 23, 2B. En este caso se puede observar rápidamente que las referencias al registro HL parecen comenzar con "2", mientras que aquellas que se refieren al registro DE comienzan con "1"; los incrementos de los registros dobles finalizan en "3" y los decrementos de registros dobles finalizan en "B".

Puedes llevar tu investigación sobre el tema más lejos y deducir que si la instrucción "Cargar BC..." es "01...", entonces la de "incrementar BC" debería ser "03", y habrás acertado.

Sin embargo, el Z80 no está completamente basado en esta idea de relaciones, interviniendo en la formación de los códigos otros factores. Pero la notación hexadecimal trae consigo el hecho de que las instrucciones del Z80 no sean un conjunto de códigos arbitrarios —formando una familia, lógicamente construida, en la que las señales binarias forman una estructura planeada de trabajos.

Todo lo expuesto hace que nos sea útil el uso de la numeración hexadecimal en la creación o uso de códigos máquina. Debo confesar que no he logrado aprenderme las tablas en hexadecimal y, por tanto, siempre acudo a ellas, aun cuando creo que conozco el equivalente hexadecimal.

Los códigos de un solo byte se encuentran fácilmente en el manual de *Sinclair,* páginas 183 a 188, y algunos de los principales códigos máquinas tienen tablas para poder trabajar con números mayores de 1 byte. Si crees que este trabajo de la conversión te supone un gran esfuerzo o te supone mucho tiempo, el programa que se analiza en el capítulo 13 es una rutina en código máquina para realizar la conversión de hexadecimal a decimal, y viceversa; esta rutina realiza la conversión instantáneamente.

He de recalcar que todas las referencias al *Manual del Spectrum* se refieren a la segunda edición de éste, que fue en 1983.

## Entrada en hexadecimal

Comprender los principios del código máquina no es lo mismo que usarlo; se necesita más que la comprensión para poder realizar programas útiles. Además de cuaderno, pluma, calculadora y manuales de consulta necesitas algún método de introducción del código máquina en el Spectrum.

Todos los libros de introducción en el código máquina presentan el listado de algún método para colocar el código máquina en la memoria RAM del Spectrum. Los hay de distinta dificultad, pero todos muestran una forma eficiente de transformar los caracteres que tecleas en tu Spectrum en bytes almacenados en la memoria. Sin embargo, he incluido aquí otro sistema.

```
   1 REM ***ENTRADA DE UN $STRIN
G EN HEXADECIMAL***
   2 INPUT "Dir. comienzo en Dec
imal?";ad
   3 LET ad1=INT (ad/256): LET a
d2=ad-256*ad1
```

```
  4 POKE USR "a",ad2: POKE USR
"A"+1,ad1: CLEAR ad-1
  10 DEF FN a(j)=CODE a$(j)-48-7
*(CODE a$(j)>=65)
  20 DEF FN b$(x)=CHR$ (INT (x/1
6)+48+7*(INT (x/16)>9))
  30 DEF FN c$(x)=CHR$ (x-16*INT
 (x/16)+48+7*((x-16*INT (x/16))>
9))
  40 LET a$="21FF00C9"
  50 IF LEN a$<>2*INT (LEN a$/2)
 THEN PRINT "Digito Hexadecimal
 sobrante.": STOP
  60 FOR j=1 TO LEN a$: IF NOT (
(a$(j)>="0" AND a$(j)<="9") OR (
a$(j)>="A" AND a$(j)<="F")) THEN
  PRINT "Error en ";j;" = ";a$(j
): STOP
  70 NEXT j
  80 LET ad=256*PEEK (USR "A"+1)
+PEEK USR "A"
  90 FOR j=1 TO LEN a$ STEP 2: P
OKE ad-1+j/2,16*FN a(j)+FN a(j+1
): NEXT j
  95 PRINT "Codigo introducido":
 STOP
 100 REM ***IMPRESION EN HEXADEC
IMAL POR PAREJAS***
 110 INPUT "Dir. comienzo en Dec
imal?";ad
 120 FOR j=ad TO ad+100
 130 LET ad1=INT (j/256): LET ad
2=j-256*ad1
 140 LET byt=PEEK j
 150 PRINT j;TAB 10;FN b$(ad1);F
N c$(ad1);FN b$(ad2);FN c$(ad2);
TAB 18;FN b$(byt);FN c$(byt)
 160 NEXT j
```

En este programa tienes que colocar en la línea 40 (a$) una cadena (conjunto de caracteres alfanuméricos) con el listado en código hexadecimal de la rutina (usando los dígitos 0 a 9 y las letras A hasta F). En las líneas 50 a 70 se comprueba que el *string* (cadena) es de una configuración correcta. Después el programa introduce, en la posición de memoria elegida (mediante POKE), los valores deseados.

La segunda parte del programa, desde la línea 100, te permite ver en pantalla los valores hexadecimales de una serie de bytes, empezando por la dirección elegida. Esta parte también muestra la dirección de cada byte en hexadecimal y decimal.

Aunque esto no es un desensamblado completo, permite un chequeo rápido de un programa entero o de cualquier otra zona de memoria. Esto se puede hacer, por supuesto, con LPRINT, si se desea.

La principal ventaja de este programa para entrada en hexadecimal reside en que el listado permanece guardado en a$. Esto posibilita su edición, cambio, depuración, etc. La mayoría de los cargadores de hexadecimal cargan el código inmediatamente en memoria, no manteniendo ninguna copia del código con la que tú pudieras trabajar.

La primera cosa que debes aprender acerca del trabajo en código máquina es que el error que a primera vista pueda parecer minúsculo, puede llegar a provocar un total desastre en el programa, la paralización de las operaciones del ordenador, etc. Si eres sensato, *siempre* guardarás tus programas (con SAVE) antes de ejecutarlos. Por el hecho de tener la rutina en un *string,* puede ser salvada con el programa BASIC sin tener que usar otro SAVE, como ocurre con los programas que son cargados directamente en memoria.

## Almacenamiento en código máquina

Este es el punto en el que hay que decidir en qué lugar de la RAM almacenaremos el código máquina.

La parte más alta de la RAM es la aconsejada por Sinclair Research (véase página 168 del Manual), y el comando "CLEAR xxxxx" ha sido incluido en el Spectrum, en parte, para poder guardar espacio de la memoria en la cual se mantenga el código mientras el programa está siendo ejecutado.

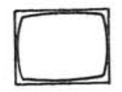
Como había dicho, la mayor parte de las rutinas que aparecen en este libro han sido colocadas en la dirección F000h —algunas veces en la E000h— y, aunque los equivalentes decimales no son fáciles de recordar, éstos serían las posiciones "61440" y "57344". Para proporcionar zonas seguras para esas dos direcciones de comienzo, debes usar la orden CLEAR en una dirección inferior en una posición, es decir, "CLEAR 61439" y "CLEAR 57343".

Otra localización segura para el código máquina es en REM, en la línea 1 de un programa BASIC.

Un programa BASIC en el Spectrum no tiene una localización fija, pero en la práctica, en el Spectrum sin expansión, el primer carácter de una línea 1 del tipo REM siempre se encuentra en la posición 23762. Sin embargo, si tú tienes un Interface 1 conectado, tendrás que encontrar la dirección que necesitas de una forma indirecta usando la variable del sistema PROG, como se explica en el capítulo 6.

REM proporciona una zona útil para una rutina corta, colocada muy próxima a un programa BASIC, como ocurre en el caso del programa Titivator del capítulo 6, el cual usa una rutina de código máquina para producir letras mayores. Un programa en esta colocación carga automáticamente y está fuera de la ejecución de cualquier otra parte del código máquina que tú puedas ejecutar (como ocurrirá en el caso del programa Titivator).

La técnica usada consiste en preparar una línea REM con el número necesario de espacios y después colocar los bytes en ella con POKE (figura 1.1).



```
1      REM 11111111111111111111
11111111111122222222222222222222
22222222222233333333333333333333
33333333333344444444444444444444
444444444444
```

*Fig. 1.1. Línea 1 REM*

Es más práctico el manejo de números en vez de usar espacios, ya que entonces puedes calcular el número total que has de dejar en REM, puesto que cada línea contiene 32 espacios.

A menudo no se listará adecuadamente la declaración REM completa al ser rellenada con POKE's, pero esto no supone ninguna diferencia respecto a la forma de funcionamiento del programa.

Si usas el programa BASIC de entrada en hexadecimal para preparar tu línea 1 REM, puedes fragmentar el programa de entrada una vez que REM esté preparada y entonces copiar en memoria la línea 1 con su aplicación final BASIC para su posterior uso, sin borrar el contenido anterior de la memoria (MERGE).

## Ensambladores y desensambladores

El problema con sencillos cargadores hexadecimales es que no proporcionan otro método de observación de programas en código máquina que una secuencia de códigos hexadecimales. En este caso, lo que se puede observar es:

```
3A 08 5C D6 20 18 06
```

en lugar de lo que sería deseable:

```
F000    3A 08 5C    LD A,(5C08)
F003    D6 20       SUB 20
F005    18 06       JR, F00C
```

o mejor aún:

```
F000    3A 08 5C    LD,A (LAS_K)
F003    D6 20       SUB 20        ; TOMA EL VALOR TECLEADO
F005    18 06       JR, PRINT
```

Obviamente, la última representación es la más detallada y, por tanto, el programa que la produce será más complicado. Para obtener una adecuada representación, realmente necesitarás comprar un programa preparado por profesionales, en cinta cassette.

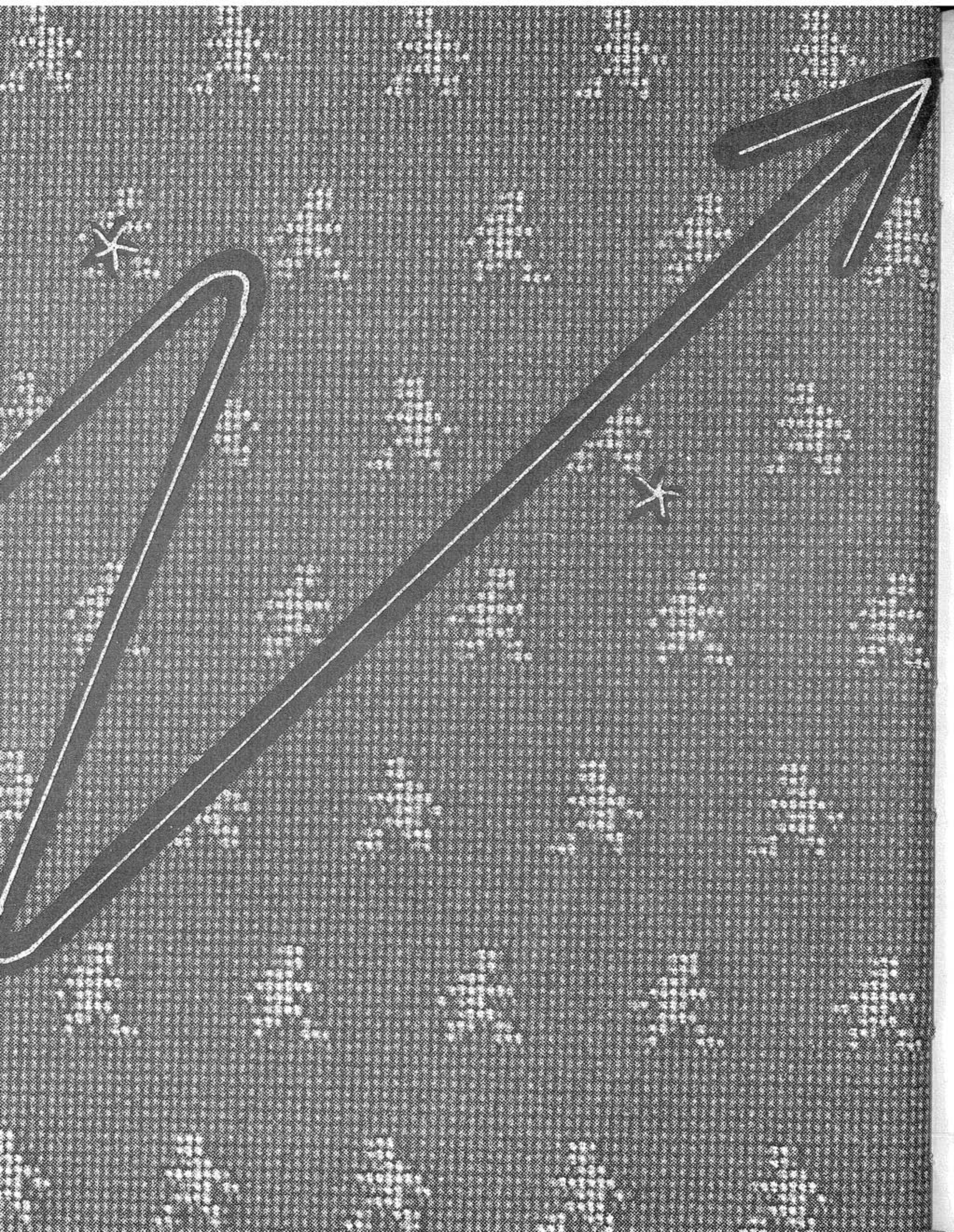
Los programas en cassette son de dos tipos básicamente, ensambladores y desensambladores, y, como es natural, hay una interrelación entre los dos cuando son cargados para su uso. Los ensambladores te permiten programar directamente en código máquina, desde el cual generarán directamente el listado del código máquina y el código

objeto. Con un desensamblador deberás teclear un código objeto, pero tendrás grandes prestaciones para la edición y representación.

La desventaja de un ensamblador radica principalmente en que resultan programas muy complicados de realizar y no muy fáciles de entender para otras personas distintas a la que lo realizó. Puede ser tan complicado teclar en código máquina y tomar todos los espacios, comas, etc., correctamente, como lo es observar el código objeto y meterlo en un desensamblador. A menudo perderás cantidad de tiempo depurando el código fuente (código máquina) en el ensamblador antes de que puedas decidir dónde comenzarás a ejecutar el programa. Por otro lado, el ensamblador tiene la ventaja de que puede utilizar normalmente etiquetas, calcula saltos de programa y hace otras tareas más útiles. Pero un buen desensamblador te permitirá depurar mucho más internamente en el código del programa, proporcionando puntos de ruptura de secuencia (BREAK), capacidad para manejar bloques de código y más posibilidades.

Hay algunos superprogramas que combinan las virtudes de ambos, pero casi siempre necesitan una impresora de 80 columnas; por eso, es más útil para los programadores profesionales que para los aficionados.

Mi preferencia personal es la del programa desensamblador —me gusta sentirme cerca del código objeto—. Pero admito que éste es un punto de vista personal y el Z80 será probablemente uno de los últimos microprocesadores en el cual esto pueda ser práctico. La generación de procesadores de 16 bit, como el 68008 en el Sinclair QL, será casi imposible manejar, excepto a través de un ensamblador.



# 2
# La memoria

El trabajo de un ordenador consiste en el movimiento de cargas eléctricas por la pastilla del microprocesador y las de las memorias. Hay dos tipos de memoria: "ROM", o memoria de sólo lectura, y "RAM", o memoria de acceso aleatorio (llamada de acceso aleatorio ya que puedes acceder a cualquier posición de memoria que tú desees dentro de la pastilla).

La memoria te la puedes imaginar, para comprender mejor su funcionamiento, como una línea muy larga de cajas numeradas (direcciones), con un "byte" de 8 bits cada una.

En el Spectrum, la línea comienza en la dirección 0000h y acaba en la FFFFh (o la 7FFFh, si se trata de un Spectrum de 16K). Puedes observar cómo los ordenadores comienzan a contar a partir de 0, en vez de a partir de 1, como lo hacemos nosotros los humanos. Esto es realmente más lógico y no es una mala costumbre comenzar a usar esta norma cuando hagamos nuestros programas; de esta manera evitaremos confusiones.

Cuando trabajes en código máquina es muy importante que seas capaz de encontrar el camino que ha recorrido tu programa a través de las memorias ROM y RAM, para lo cual un mapa o plano de su configuración es muy útil (mapa de memoria).

El mapa de memoria de la figura 2.1 no es muy completo. Podrás encontrar otras versiones en el manual de *Sinclair* y en la mayoría de los libros que abordan el tema del código máquina en el Spectrum. La versión que usamos en este libro ha sido confeccionada para cubrir nuestras necesidades y, por tanto, no utiliza detalles que no nos resulten importantes para el uso que vamos a hacer aquí de la memoria.

## El mapa de memoria

Comenzando desde la parte inferior de ésta, la zona de memoria desde la dirección 0000h hasta la de 3FFFh está ocupada por la ROM. Esta es la zona en la que reside toda

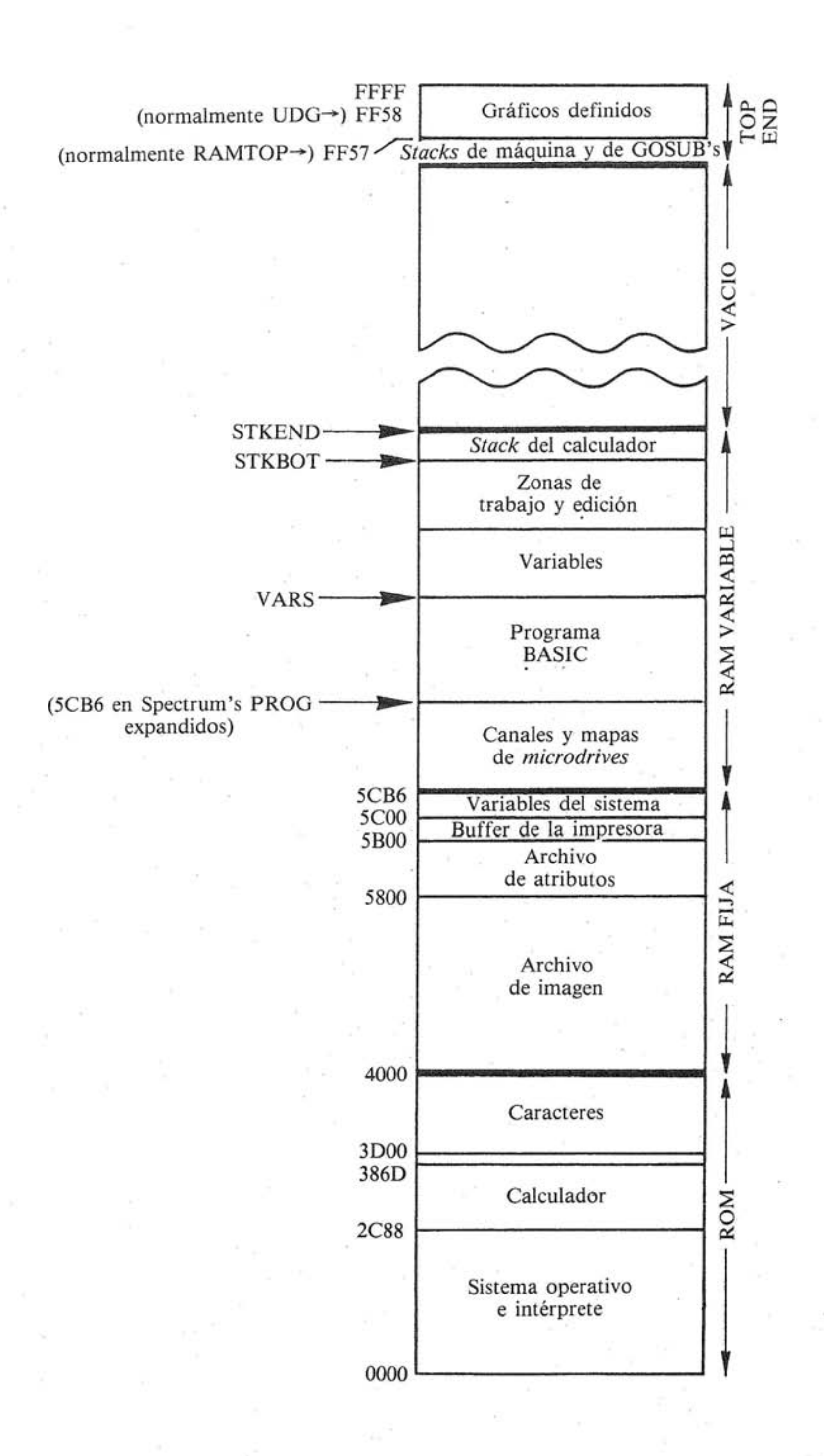



*Fig. 2.1. Mapa de Memoria del Spectrum*

la capacidad de operación del Spectrum y no es modificable, aunque su contenido se puede leer y, por tanto, estudiar. Es usada en la mayoría de los programas, tanto el BASIC (en el cual se usa siempre) como en el código máquina, cuando su utilización puede ser una ayuda. Todos los libros tienen escrito en sus páginas algo sobre la ROM, y nosotros también lo haremos más tarde.

Después de la ROM, el contenido de la memoria se hace más aleatorio. Esta zona de memoria puede ser llenada con información por el operador del ordenador, por una cinta cassette o un *microdrive,* pero, en todos los casos, estas zonas de memoria perderán dicha información al apagar el ordenador, como desgraciadamente muchos de nosotros ya sabemos muy bien.

Lo primero que está situado sobre la ROM es la zona denominada "RAM fija". Esta zona contiene: partes que utiliza la ROM, en unas direcciones fijas; una gran zona para mantener el fichero de imagen y los atributos de color para la televisión; una zona pequeña donde se ensamblan los datos para la impresora y la parte más importante, la cual contiene todas las direcciones usadas por la ROM (o por nosotros), cuando manejamos el Spectrum y que se denomina "variables del sistema".

Detrás de esta zona, el contenido de la RAM se hace aún más difuso y se le denomina "RAM flotante". Esta zona no tiene una longitud predeterminada; no obstante, sus direcciones se procesan siempre en la ROM y después se mantienen en la zona de variables del sistema.

Hay zonas que operan con la información del canal y con el *microdrive.* Tras ellas viene la zona en la que se almacenan todos los programas BASIC que hacemos, con las variables que usan dichos programas. Por último, están las zonas que usa el Spectrum para trabajar con programas en BASIC, como son la zona de trabajo *(work space),* la zona de edición y la zona del *stack* de cálculo.

Detrás queda una zona sobrante que puede ser grande o pequeña, dependiendo de cómo sean las zonas que ocupan los programas en BASIC y sus variables. Puesto que hay un espacio sobrante, y por tanto libre, éste se puede llenar con otros datos o códigos máquina. Esta es la zona de la RAM que nosotros usaremos más a menudo para escribir y ejecutar nuestras rutinas en código máquina.

Limitando la RAM por arriba, está lo que en el capítulo 1 llamamos *top end* o parte más alta. Esta parte normalmente se rellena desde la zona más alta hacia abajo. En primer lugar, hay una parte para contener los gráficos diseñados por el usuario, que normalmente está en la dirección indicada por la variable del sistema UDG, pero tú puedes cambiar cuando quieras el valor de la UDG y hacer que apunte hacia cualquier otro lugar (lo cual puede serte útil). Por debajo de la UDG se encuentra una dirección llamada RAMTOP (parte alta de la RAM), con algunos *stacks* más (zonas para almacenamiento de números) por debajo de ella. Cualquier operación RUN, CLEAR o NEW normalmente limpiarán la RAM sólo hasta la dirección RAMTOP (solamente cuando se apaga el ordenador, se limpia por detrás de esta dirección); por tanto, moviendo el valor de RAMTOP hacia abajo puedes salvar a una parte de la memoria de ser escrita de nuevo por una operación BASIC (véase página 132 del *Manual del Spectrum*).

## La ROM

La parte más interesante de la memoria es, sin lugar a dudas, la ROM. Todo lo que hace el Spectrum cuando escribe o ejecuta un programa BASIC se realiza a través de la ROM. Tiene un programa para cada cosa. En realidad, se puede decir que la ROM *es* el Spectrum.

La totalidad de esos programas están escritos en código máquina del Z80 y, formando parte de los programas principales, hay llamadas a subrutinas del sistema que se usarán cuando sean necesarias para realizar determinadas tareas. Estas son las posibilidades que usaremos después, ya que pueden hacer las mismas tareas para nosotros, y con ello nos ahorramos una gran cantidad de problemas.

Las rutinas que nos interesarán para la realización de las aplicaciones que hay en el libro están listadas en el apéndice B, pero hay muchas rutinas más.

Ahora es el momento en que puedes apreciar las ventajas que supone el tener una aplicación que te permita manejar cualquiera de las rutinas del sistema, es decir, el ya mencionado desensamblaje de la ROM del Spectrum, permitiéndote posicionar en la memoria RAM la rutina en la que estás interesado y el trabajar sobre su listado para ver si puedes usarla como una subrutina de tus propios programas. Hay partes de esos listados que no te servirán según están, pero puedes esforzarte y probar modificando en esas zonas de la rutina para que te sean útiles.

Para mostrarte cómo se pueden usar las rutinas de la ROM para realizar cosas para las que no fueron creadas originalmente, he aquí un pequeño programa que proporciona una forma automática de guardar "bytes". Este programa puede serte útil si tienes un programa en BASIC que siempre trabaje acompañado de un bloque de datos. Los datos pueden ser el contenido de una pantalla o una rutina en código máquina; puede ser cualquier cosa, ya que se guardará siempre como un conjunto de bytes.

Este programa compila una etiqueta *(label)* perteneciente al programa BASIC, es decir, puede calcular una nueva etiqueta, como si fuera un dato o un número índice, cada vez que es almacenado (lo cual puede ser útil también para ti).

**Almacenador («Saver»)**

```
   5    DIM w$(10)
   6    DEF FN a$(x)=CHR$ INT (x
/256)
   7    DEF FN b$(x)=CHR$ (x-256
*CODE FN a$(x))
  10    INPUT "Etiqueta:";w$: PR
INT AT 8,0;"Etiqueta de cabecera
: ";w$
  20    INPUT "Dir. comienzo en
Decimal:";st
  30 INPUT "Dir. final:";fi
```

```
  40     LET le=fi-st
  50     PRINT AT 10,0;"Almacenam
iento bytes desde ";st," hasta "
;fi
  60     LET z$=CHR$ 3+w$+FN b$(l
e)+FN a$(le)+FN b$(st)+FN a$(st)
+CHR$ 0+CHR$ 0+CHR$ 33+CHR$ 82+C
HR$ 0+CHR$ 229+CHR$ 33+FN b$(st)
+FN a$(st)+CHR$ 229+CHR$ 221+CHR
$ 33+CHR$ 0+CHR$ 91+CHR$ 195+CHR
$ 132+CHR$ 9
  70     FOR j=1 TO LEN z$: POKE
23295+j,CODE z$(j): NEXT j
  80     RANDOMIZE USR 23313
```

Debido a que es un programa de demostración, utiliza "W$" (variable con la etiqueta), "st" (variable en la que se encuentra la dirección de comienzo) y "fi" (variable con la dirección final) como entradas (INPUTs), pero lo normal es que tú lo prepares para que estas variables sean rellenadas o calculadas desde tu programa principal.

Cada uno debe saber, cuando carga un programa al Spectrum, que hay una especie de "mini-programa", el cual es cargado antes que el programa principal. Este "mini-programa", conocido como "cabecera", al igual que la etiqueta (nombre) del programa, que es escrita en la pantalla, contiene una información importante acerca del programa principal, la cual permite al Spectrum realizar la carga adecuadamente.

Por tanto, antes de que podamos saber cómo funciona el programa almacenador *(Saver)*, antes listado, es necesario que tengamos claro cómo es formada la "cabecera".

Consiste en 17 bytes ordenados de la siguiente manera:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 3 | | | | ETIQUETA | | | | | | le | | | st 0 | 0 | |

El byte 0 se codifica según el tipo de datos a ser guardado: "0" si es un programa en BASIC, "1" y "2" si se trata de *arrays* (matrices) numéricos o de caracteres, "3" si es un bloque de bytes. Los 10 bytes siguientes contienen la etiqueta del programa que es normalmente introducida a mano (tecleada), pero aquí la puedes encontrar en la variable "W$". La "cabecera" acaba con tres parejas de bytes; la primera pareja contiene la longitud del bloque a guardar (la variable "le"), la siguiente pareja contiene la dirección de comienzo del bloque (variable "st") y la pareja final, la cual, en caso de un bloque de código como el visto, está rellena de ceros.

Toda esta cabecera es ensamblada en la primera zona de la variable "Z$", en la línea 60 del programa, la cual es almacenada (con POKE) al comienzo del *buffer* para la impresora que está en la dirección 5B00h, en la línea 70 del programa almacenador. Se ha elegido el *buffer* de la impresora ya que no es utilizado en este programa, y su empleo

nos evita tener que utilizar, para otro uso distinto al que había sido reservada, otra zona de memoria RAM.

La segunda parte de la línea 60, almacenada en las direcciones desde la 5B11h en adelante, consiste en algunas instrucciones en código máquina. Aquí se muestra la apariencia que toma cuando ha sido puesta en el *buffer* de la impresora:

**Código del almacenador**

```
5B11            10       ORG  #5B11
5B11 215200     20       LD   HL,#0052
5B14 E5         30       PUSH HL
5B15 2100F0     40       LD   HL,#F000
5B18 E5         50       PUSH HL
5B19 DD21005B   60       LD   IX,#5B00
5B1D C38409     70       JP   #0984
```

Como puedes ver, intervienen aquí tres direcciones, dos de ellas almacenadas en el *stack* y una almacenada en el registro IX, seguida de un salto a una rutina en la ROM. Para saber lo que hacen, necesitamos observar la rutina de almacenamiento residente en la ROM.

**Comienzo de la rutina SA-CONTROL**

```
0970            10       ORG  #0970
0970 E5         20       PUSH HL
0971 3EFD       30       LD   A,#FD
0973 CD0116     40       CALL #1601    ;Abre la parte baja de la pantalla
0976 AF         50       XOR  A
0977 11A109     60       LD   DE,#09A1
097A CD0A0C     70       CALL #0C0A    ;Imprime 'Start Tape'
097D DDCB02EE   80       SET  5,(IX+2)
0981 CDD415     90       CALL #15D4
0984 DDE5      100       PUSH IX
0986 111100    110       LD   DE,#0011  ;Longitud de la cabecera
0989 AF        120       XOR  A
098A CDC204    130       CALL #04C2    ;Rutina principal de almacenamiento
098D DDE1      140       POP  IX
098F 0632      150       LD   B,#32
0991 76        160       HALT ;Parada de 1 segundo
0992 10FD      170       DJNZ #0991
0994 DD5E0B    180       LD   E,(IX+#0B)
0997 DD560C    190       LD   D,(IX+#0C)
099A 3EFF      200       LD   A,#FF
099C DDE1      210       POP  IX
099E C3C204    220       JP   #04C2
```

No es necesario que entremos en detalles, pero a través de las notas aclaratorias en la rutina debe de quedar clara, al menos, una ligera idea sobre cómo funciona la rutina. Antes de que comience a ejecutarse la rutina, el par de registro HL debe de contener la dirección de comienzo del bloque a ser guardado (SAVE), y el registro índice IX debe contener la dirección de la información de "cabecera". La rutina comienza introducien-

do el par de registros HL en el *stack* (PUSH) y, después de esto, imprime el mensaje "Corra la cinta...", en la parte baja de la pantalla, antes de esperar a que pulses una tecla del teclado. Cuando la pulsas, la rutina pasa el registro índice IX al *stack* (PUSH) y carga el par de registros DE con la longitud fija del encabezamiento o "cabecera", que como vimos anteriormente es de 17d = 11h.

Después llama (CALL) a la subrutina principal, denominada "SA_BYTES", que comienza en la dirección 04C2h. Esta subrutina guardará el número de bytes indicado en el registro DE, comenzando por el que está en la dirección indicada por IX.

Una vez que esto se ha realizado, hay una pausa. La rutina carga entonces el par de registros DE con la longitud del bloque de datos usando el registro IX para elegir la información de "cabecera" que nos interesa (IX apunta a la dirección de comienzo de la "cabecera"). La rutina mete entonces en el registro IX la última dirección del *stack* (mediante un POP), la cual fue introducida en él, procedente del registro HL, por medio de una instrucción PUSH; ésta es la dirección de comienzo del bloque a ser guardado o salvado en cinta (SAVE). Por tanto, el registro IX contendrá ahora la dirección de comienzo y la rutina estará ya preparada para almacenar el bloque, saltando de nuevo a la dirección 04C2h.

Observa de nuevo las tres direcciones que hay en el *buffer* de la impresora. La primera que fue introducida en el *stack* será la última en salir de éste y será la dirección de retorno para la instrucción retorno (RET) de la segunda llamada a la subrutina "SA_BYTES". Esta segunda llamada, como recordarás, se hizo en la línea 099Eh por medio de un salto (JP). La dirección de retorno contiene, de hecho, otra instrucción RET. Esta ha sido reservada para que regresemos al BASIC cuando hayamos completado la operación en curso.

La siguiente en el *stack* es la dirección de comienzo del bloque de datos, que fue calculada en el programa BASIC por FNa(x) y FNb(x). La tercera es la dirección de comienzo del *buffer* de impresora que está preparado en el registro IX.

Si observas y examinas la dirección 0984h, dirección de ROM a la que saltamos en la ejecución de la rutina original de almacenamiento residente en la ROM, verás que nuestra rutina ha saltado todos los pasos en los que se esperaba para que fuera pulsada una tecla. A pesar de esto, todas las direcciones necesarias están ya preparadas en el *stack* y en IX, por lo que el resto de la rutina en ROM puede continuar como lo habíamos planeado.

Para finalizar esta demostración, a continuación se da el listado tal y como aparecerá en un programa. Tú guardas la información por medio de la instrucción "GO TO 1100" (esto almacenará el programa seguido por los bytes que hayas especificado con CODE). Cuando procedás a cargarlo (LOAD), te cargará el programa y comenzará a ejecutarlo a partir de la línea 1000, en la cual se ordena cargar inmediatamente los bytes.

Tendrás que ejecutar una instrucción del tipo CLEAR "bytes — 1" antes de usarla.

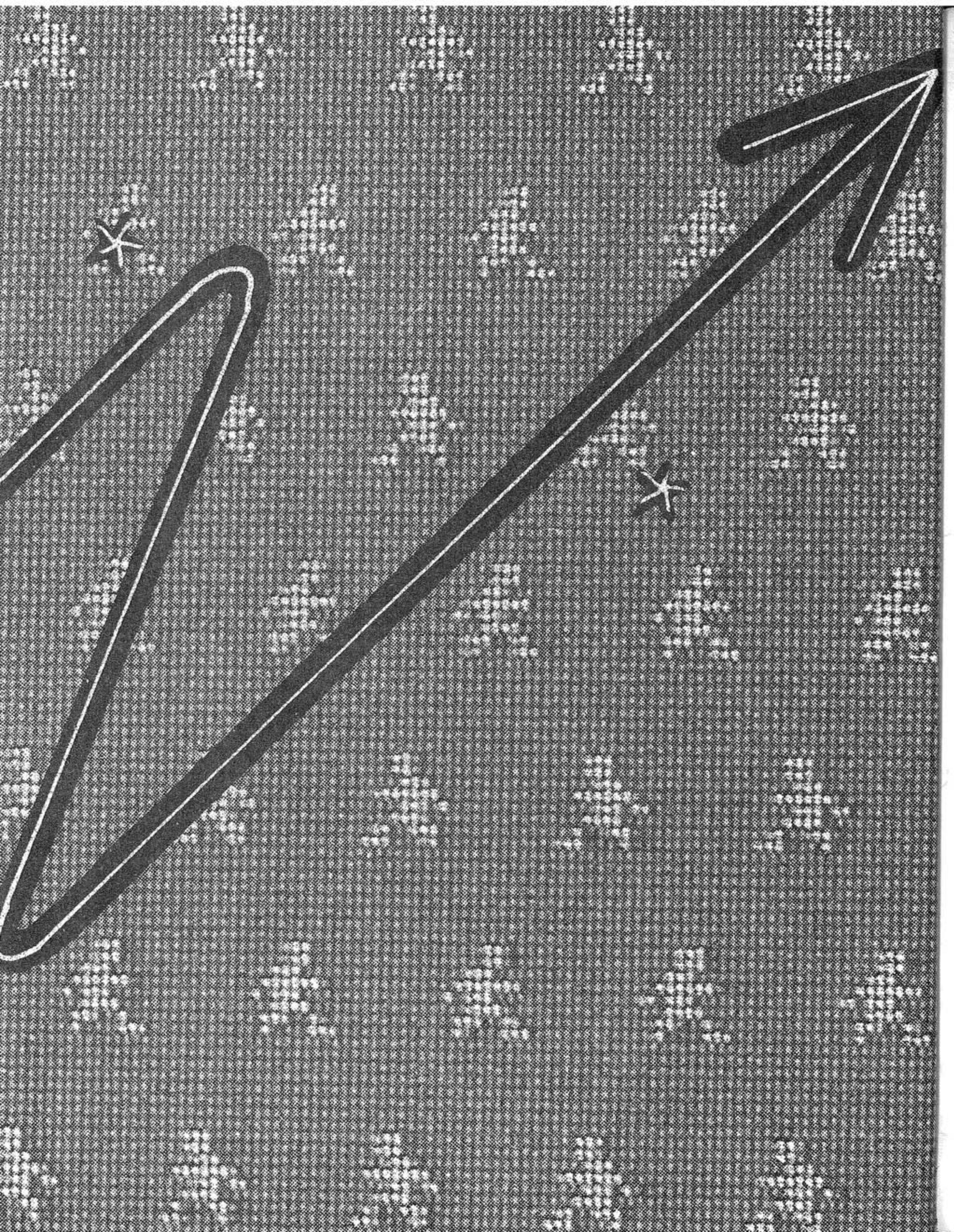
**Ejemplo de almacenamiento**

```
1000  LOAD ""CODE
1010  GO TO 10
```

```
1100   SAVE "Cualquiercosa" LINE
 1000
1110   DIM w$(10)
1120   DEF FN a$(x)=CHR$ INT (x/
256)
1130   DEF FN b$(x)=CHR$ (x-256*
CODE FN a$(x))
1140 LET w$="CualquiercosaCODE "
1150  LET st=64256: LET le=1280
1160   LET z$=CHR$ 3+w$+FN b$(le
)+FN a$(le)+FN b$(st)+FN a$(st)+
CHR$ 0+CHR$ 0+CHR$ 33+CHR$ 62+CH
R$ 0+CHR$ 229+CHR$ 33+FN b$(st)+
FN a$(st)+CHR$ 229+CHR$ 221+CHR$
 33+CHR$ 0+CHR$ 91+CHR$ 195+CHR$
 132+CHR$ 9
1170   FOR j=1 TO LEN z$: POKE 2
3295+j,CODE z$(j): NEXT j
1180   RANDOMIZE USR 23313
```

Esto es un ejemplo de la manera en la que puedes dedicar la ROM para tus propósitos.

Confieso que es duro enfrentarse con algo tan complicado. Puede resultar complicado el estudio del programa incluso con la ayuda de un listado procedente del desensamblaje de la ROM y algunas veces la secuencia prueba-error se convierte en la de error-prueba.



# 3
# Realización de caracteres más grandes

## Imprimir un texto

El Sinclair Spectrum utiliza uno de los formatos más atractivos y legibles en el campo de los ordenadores. Utiliza una matriz de 8 × 8 bits para producir las letras, los cuales están almacenados como ocho bytes por letra en la ROM de caracteres, que ocupa un espacio que va desde la dirección 3D00h hasta la 3FFFh.

Este es un caso casi exagerado de bits para la impresión de caracteres. Muchas de las impresoras matriciales actualmente en el mercado sólo utilizan matrices de 5 × 7, lo que significa que han de utilizar recursos especiales para mover la posición de impresión sobre el papel, como puede ser el dar un espacio entre letras. Además, las minúsculas no pueden tener verdaderas letras "descendentes".

Descendentes son las letras con apéndices tales como la "p", "q" o "y", las cuales normalmente caen por debajo de la línea de impresión.

En una matriz de 5 × 7 es complicado obtener esto, por lo que las firmas que las fabrican utilizan una pequeña trampa en sus impresoras, como se ve en la figura 3.1. Esto da como resultado una difícil observación y legibilidad del texto.

```
To get a fuller reply we need
rest of the original sentence
inserting a space). It is not
```

*Fig. 3.1*

En el Spectrum hay letras "descendentes" como las mencionadas y la anchura de ocho bits significa que los caracteres pueden ser impresos en diferentes posiciones, manteniendo los espacios entre las letras. Observa la figura 3.2, en la que una línea ha sido subrayada a través de los bits más bajos de cada carácter, de forma que los apéndices de las letras q, p, y, j cortan la línea inferior. Los apéndices ascendentes de las letras t, h, f, k, l tocan a la línea superior.




*Fig. 3.2*

Antes de que comencemos a ver cómo podemos utilizar y modificar el conjunto de caracteres del Spectrum, tal vez fuera conveniente echar una rápida ojeada a cómo el Spectrum realiza la impresión de dichos caracteres.

La forma de realizar casi todas las posibles impresiones del Spectrum es mediante la instrucción "Restart 10", en código del Z80. Este código de operación "D7" se usa para introducirse en la rutina principal de escritura de la ROM (ROM PRINT ROUTINE). Esta rutina controla todas las operaciones de impresión en el Spectrum, incluyendo la fijación del color, posición de impresión y otros atributos (véase apéndice B).

La impresión de los números se realiza de otra forma. Esta requiere colocar el valor del número en el *stack* del calculador, en formato de cinco "bytes" en coma flotante, desde donde puede ser sacado e impreso con punto decimal o en forma exponencial. Esto se realiza por medio de una rutina ROM que comienza en la dirección 2DE3h. Sin embargo, para nuestros propósitos actuales, la instrucción "RST # 10" es la que usaremos.

Para usar esta instrucción, primero has de decidir qué clase de impresión vas a hacer. Normalmente, cuando trabajamos con una operación USR, el Spectrum estará en modo de entrada (INPUT *mode*) e imprimirá en la parte inferior de la pantalla. Para hacer que el Spectrum imprima en la zona de ejecución de la pantalla debes cubrir el canal "S", usando para ello las instrucciones "LD A,02: CALL 1601" (1601h es la dirección de la rutina de "abrir canal").

Por tanto, para imprimir la letra "A" en esta zona de pantalla necesitarás la rutina:

**Imprimir "A"**

```
F000                ORG  #F000
F000 3E02           LD   A,2
F002 CD0116         CALL #1601 ;Abre canal pantalla
F005 3E41           LD   A,#41
F007 D7             RST  #10 ;Imprime letra "A"
F008 C9             RET
```

Si cambias el valor "02" que hay en la posición de memoria F001h por el valor "03", la "A" será enviada a la impresora en vez de a la pantalla. Si lo cambias por el valor "01", la letra "A" irá a la zona de entrada (INPUT *área*), pero no siempre puedes verlo tú, ya que esta zona normalmente se limpia tan pronto como la operación se ha completado. Puedes mantener la "A" en la pantalla usando los comandos BASIC, "RANDOMIZE USR 61440: PAUSE 0".

Para imprimir la "A" en una determinada posición, en un determinado color, hemos de incorporar los códigos de control adecuados (los encontrarás en la página 183 del *Manual del Spectrum*). Para imprimir una "A" verde en un fondo amarillo, en la línea 10, en la decimosexta columna, podemos hacer lo siguiente:

**Imprimir un «string»**

```
F000              10    ORG  #F000
F000 3E02         20    LD   A,02
F002 CD0116       30    CALL #1601 ;Abre canal pantalla
F005 3E16         40    LD   A,#16
F007 D7           50    RST  #10 ;Activa control "AT"
F008 3E0A         60    LD   A,#0A
F00A D7           70    RST  #10 ;Pone cursor en linea 10
F00B 3E10         80    LD   A,#10
F00D D7           90    RST  #10 ;Pone cursor en columna 16
F00E 3E10        100    LD   A,#10
F010 D7          110    RST  #10 ;Control "INK"
F011 3E04        120    LD   A,#04
F013 D7          130    RST  #10 ;Imprime en verde
F014 3E11        140    LD   A,#11
F016 D7          150    RST  #10 ;Control "PAPER"
F017 3E06        160    LD   A,#06
F019 D7          170    RST  #10 ;Fondo amarillo
F01A 3E41        180    LD   A,#41
F01C D7          190    RST  #10 ;Letra "A"
F01D C9          200    RET
```

Todo esto está muy bien una vez que está en marcha, pero es muy engorroso y no usaremos este sistema para imprimir muchas instrucciones, o un texto, en un programa.

Afortunadamente, Sinclair Research ha incorporado una subrutina para la impresión de cadenas de caracteres *(strings)* en la ROM, en la posición 203Ch, la cual nos libera de la mayoría de los problemas que plantea la impresión. Necesitas tener la dirección de la cadena de caracteres en el registro DE, y su longitud en el registro BC, antes de llamar a la dirección 203Ch. La subrutina es, principalmente, una forma de pasadas sucesivas por el *string,* usando "RST 10" para imprimir un carácter en cada pasada.

Nuestra "A" verde, quedará, por tanto:

**"A" verde**

```
F000              10    ORG  #F000
F000 160A100A     20    DEFB #16,#0A,#10,10,#04,#11,#06,#41
F008 1100F0       30    LD   DE,#F000
F00B 010800       40    LD   BC,#0008
F00E CD3C20       50    CALL #203C
F011 C9           60    RET
```

Aquí permanecen los mismos códigos para el fondo, posición de impresión, etc., pero ahora están todos agrupados en la dirección F000h. He omitido la selección del canal para evitar confusiones, pero tú puedes incorporarla cada vez que la necesites para reencaminar la impresión a la zona de ejecución de la pantalla.

El código es ahora más compacto, pero puede tomar un aspecto más desordenado cuando haya un lote de mensajes para ser impresos en un programa en código máquina.

El primer paso es añadir la longitud de la cadena de caracteres al comienzo de los datos, de forma que el DEFB (definición de bytes) se hace:

```
F000              10        ORG  #F000
F000 08160A10     20        DEFB #08,#16,#0A,#10,10,#04,#11,#06,#41
```

Este número puede ser recuperado por la nueva rutina y cargado en el registro BC al comienzo de las operaciones.

```
F000              10        ORG  #F000
F000 08160A10     20        DEFB #08,#16,#0A,#10,#10,#04,#11,#06,#41
F009 3E02         30        LD   A,2
F00B CD0116       40        CALL #1601
F00E 1100F0       50        LD   DE,#F000
F011 1A           60        LD   A,(DE) ;Pasa primer byte direccionado
                  70 ;                   por reg. DE al reg. BC
F012 4F           80        LD   C,A
F013 0600         90        LD   B ,00
F015 13           100       INC  DE     ;Puntero de la cadena
F016 CD3C20       110       CALL #203C
F019 C9           120       RET
```

Una vez que se ha impreso la cadena, puedes llamar de nuevo a la dirección 203Ch, esta vez para imprimir otra cadena de caracteres, la cual utiliza cualquier color o cualquier atributo a su aspecto normal, de forma que los actuales atributos no se mantendrán para la siguiente impresión, la cual requerirá que algunas cosas sean diferentes. Esta es la sección final para realizar una impresión "casera".

```
F012              10        ORG  #F012
F012 010C00       20        LD   BC,#000C ;Long. cadena restauracion
F015 111CF0       30        LD   DE,#F01C ;Dir. cadena restauracion
F018 CD3C20       40        CALL #203C
F01B C9           50        RET
F01C 10091108     60        DEFB #10,#09,#11,#08,#12,#00,#13,#00,#14,#00,#15,#00

F012              70        ENT  #F012
```

Tú puedes, por supuesto, seleccionar los atributos normales que deseas cuando vayas a realizar la impresión de la cadena.

Ahora, cuando llames a la subrutina, tendrás solamente que especificar la dirección; la subrutina leerá su propia longitud para el control del lazo. Tú puedes agrupar todos los mensajes en un bloque.

Advierte, de paso, que has de volver a fijar el valor del registro BC a F012h, ya que este registro es cambiado de valor por la rutina de dirección 203Ch.

## Dilatación de caracteres

Este nuevo capítulo es más interesante para los propietarios del Spectrum que ya han utilizado la cassette "Horizons" y que, al hacerlo, se han preguntado cómo podrían imprimir con el Spectrum caracteres con todas las formas y tamaños, tal y como lo hace la cassette.

De hecho, la rutina en código máquina "Psion" para la dilatación de letras es bastante buena: Como dicha rutina forma parte de los accesorios originales del Spectrum, es posible sacarla de la cinta al Spectrum. No es difícil escribir un pequeño programa en BASIC para hacer que la rutina aumente los caracteres tanto como tú desees. Sin embargo, la rutina "Psion" es un poco larga y complicada y tiene, como posible fallo, que es demasiado buena. Las posibilidades que ofrece son, en algunas ocasiones, demasiadas y muy complicadas.

Para usos prácticos, dentro de la trama de un programa, encuentro que la posibilidad más útil es la de duplicar la longitud de un carácter dándole una apariencia agradable; aun así es demasiado pequeño como para representar una buena línea de impresión (16 caracteres). Pero hay muchas posibilidades útiles. Vamos a ver cómo pueden ser realizadas.





Para realizar letras de doble tamaño hemos de crear un bloque desarrollado en 4 bits, es decir, donde sólo había un bit en el carácter original ahora habrá un grupo de 4 bits. Esto significa que, para colocar el carácter así aumentado, hemos de utilizar el espacio que antes ocupaban cuatro caracteres normales. La figura 3.3 muestra cómo se juntan cuatro caracteres de dimensiones normales formando el espacio que utiliza un carácter duplicado en tamaño. En la misma figura, observando las representaciones en negativo *(inverse printing),* puedes ver claramente cómo el carácter aumentado está compuesto de cuatro caracteres gráficos de tamaño normal.

AC
BD

A C
B D
Normal

Inverse

*Fig. 3.3. Grandes caracteres gráficos compuestos de cuatro caracteres normales*

Esta es la manera de trabajar que tiene la rutina. Por cada carácter aumentado que imprimimos, producimos un conjunto de cuatro nuevos caracteres gráficos en las posi-

ciones "UDG" A, C, B y D, después las imprimimos como se ha visto. Para ahorrarnos tiempo nos referiremos a las posiciones "UDG" por los nombres de las letras que normalmente encontraríamos ahí.

Para producir esos caracteres gráficos necesitamos una rutina en código máquina. En principio esta rutina trabajará de una forma sencilla.

En primer lugar, seleccionaremos y la buscaremos en el conjunto de caracteres del Spectrum. Cada carácter está formado por ocho bytes en dicho conjunto de caracteres, cada uno de los cuales codifica los puntos de impresión de una línea en la pantalla. Un bit "1" equivale a un punto negro *(black pixel)* y un bit "0" corresponde a un punto blanco *(white pixel)*. Para nuestros propósitos, trabajaremos con trozos de caracteres de cuatro bytes (en primer lugar los cuatro de la parte superior y, después, los cuatro de la parte inferior del carácter).

En primer lugar, pondremos el primer byte del carácter escogido en el registro A. Es decir, ponemos en A la línea superior del carácter. Desafortunadamente, la línea superior del carácter está formada solamente por ceros, por lo que cogeremos, para la explicación, una línea arbitraria (figura 3.4), la cual muestra los movimientos de bits de la línea de una forma más clara. En la figura hemos tomado la segunda línea, desplazada dos bits a la derecha. Después almacenamos el bit en la posición 0 en el bit de acarreo *(carry bit)*, ejecutando una instrucción "RRA" (rotación a la derecha del acumulador).

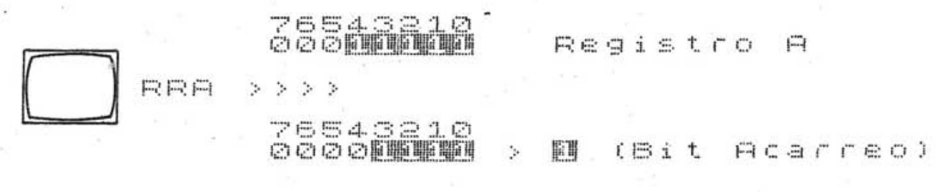



*Fig. 3.4*

Escogemos el par de registros DE para guardar los dos nuevos bytes, los cuales se generarán a partir del byte almacenado en el registro A; desplazamos el bit de acarreo a la posición 7 del registro D con una instrucción "RR D" (figura 3.5). Esto, por supuesto, supone que el bit cero que estaba en la posición 0 del registro D, pase al bit de acarreo, por lo que lo recuperamos de nuevo, pasándolo al registro E, ejecutando la instrucción "RR E" (véase figura 3.5).

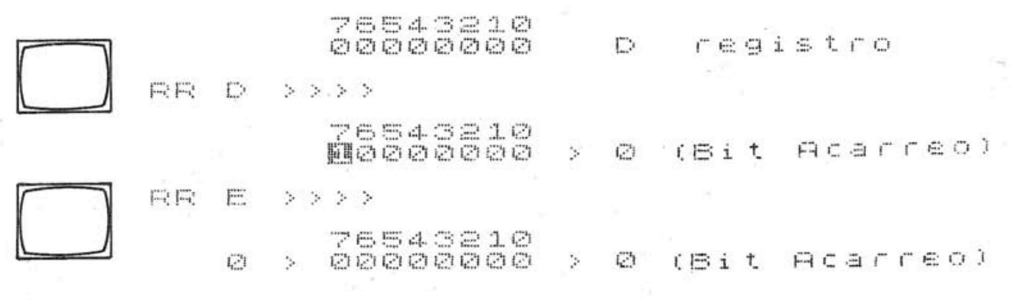



*Fig. 3.5*

Como el nuevo carácter será el doble de grueso que el antiguo, necesitamos doblar el nuevo bit obtenido del registro A. Esto se puede realizar usando la instrucción "SRA D" (desplazamiento aritmético a la derecha), que desplaza todos los bits un lugar a la derecha, pero además copia en el bit 7 el valor del bit previamente situado aquí, que es lo que nosotros queríamos. Para completar la operación, recuperamos el bit 0, que con esta operación hemos situado en el bit de acarreo, haciendo de nuevo una "RR E" (figura 3.6).

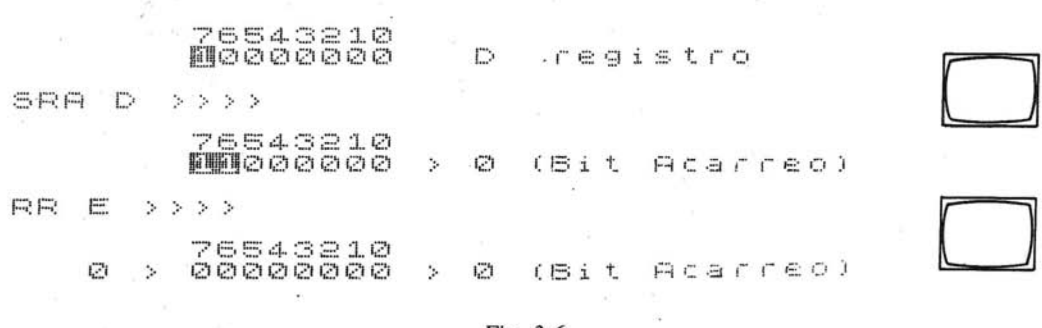



*Fig. 3.6*

Puedes ver que, después de hacer esto mismo ocho veces, habremos copiado todos los bits de A como bits dobles en los registros D y E. Todo lo que ahora queda es copiar los registros D y E en los bytes apropiados de UDG "A" y "C".

Como nuestros nuevos caracteres serán también dos veces más gruesos horizontalmente, como los son verticalmente, copiaremos los registros D y E por segunda vez, en los dos próximos bytes de UDG "A" y "C"; éstos formarán la segunda línea del nuevo carácter en la pantalla (figura 3.7).




*Fig. 3.7*

Después de ejecutar cuatro operaciones como las vistas, habremos finalizado la mitad superior del carácter y habremos completado todos los bytes de UDG "A" y "C". Pero para continuar con el programa, ejecutamos esas operaciones para UDG "B" y "D", y completamos con la mitad inferior del carácter usando la misma técnica.

Ahora mostramos el listado completo. Las operaciones entre F000h y F00Fh tratan la espera y captación del carácter introducido en el teclado desde la variable de sistema LAST_K y la ejecución de la dirección en el conjunto de caracteres para el carácter este, que se encuentra en HL. El resto de la rutina genera los cuatro nuevos caracteres gráficos. Observa que, cuando encuentras la dirección para los caracteres UDG, lo haces indirectamente a través de la variable del sistema UDG en 5C7Bh. Esto te permite seleccionar una dirección diferente, si quieres hacerlo.

**Letras de doble tamaño**

```
F000                ORG  #F000
F000 210000         LD   HL,0000 ;Pone a cero HL
F003 3A085C         LD   A,(#5C08) ;Pasa a reg. A la ultima tecla
F006 D620           SUB  #20
F008 6F             LD   L,A       ;Mete en HL valor codigo de tecla
F009 29             ADD  HL,HL     ;Multiplica por 8
F00A 29             ADD  HL,HL
F00B 29             ADD  HL,HL
F00C 01003D         LD   BC,#3D00     ;Dir. comienzo de alfabeto
F00F 09             ADD  HL,BC
F010 DD2A7B5C       LD   IX,(#5C7B) ;Dir. de UDG.
F014 0E08           LD   C,#08
F016 7E             LD   A,(HL)
F017 0608           LD   B,#08
F019 1F             RRA
F01A CB1A           RR   D
F01C CB1B           RR   E
F01E CB2A           SRA  D
F020 CB1B           RR   E
F022 10F5           DJNZ #F019
F024 DD7200         LD   (IX),D
F027 DD7201         LD   (IX+01),D
F02A DD7310         LD   (IX+#10),E
F02D DD7311         LD   (IX+#11),E
F030 23             INC  HL
F031 DD23           INC  IX
F033 DD23           INC  IX
F035 0D             DEC   C
F036 20DE           JR   NZ,#F016
F038 C9             RET
```

Ahora puedes observar los dos programas en BASIC que vienen a continuación; estos dos pequeños programas utilizan las rutinas para duplicar el tamaño de las letras. Uno de ellos te permite teclear las letras que desees introducirle como en una máquina de escribir. El segundo te muestra una impresión de una cadena de letras en tamaño doble.

Observa que ambos programas introducen la letra a duplicar en la variable del sistema LAST_K; el programa 1 desde el teclado, y el programa 2 introduciéndole directamente en la variable del sistema con instrucciones POKE. LAST_K es un punto muy bueno de acceso para las técnicas de impresión que necesitan una comunicación *(interface)* entre un programa BASIC y el código máquina. Es una de las formas más fáciles de recoger un carácter, aunque parezca una forma un poco indirecta.

**Tecleado de letras en tamaño doble**

```
100 LET x=0: LET y=0
110 PAUSE 0
120 IF CODE INKEY$=13 THEN  LET
x=0: LET y=y+2: GO TO 110
130 RANDOMIZE USR 61440
140 PRINT  BRIGHT 1;AT Y,2*x;"A
C";AT y+1,2*x;"BD"
150 LET x=x+1
160 GO TO 110
```

**Impresión de una cadena de letras en doble tamaño**

```
 10 INPUT w$
110 FOR j=1 TO LEN w$
120 POKE 23560,CODE w$(j)
130 RANDOMIZE USR 61440
140 PRINT  BRIGHT 1;AT y,2*x;"A
C";AT y+1,2*x;"BD"
150 LET x=x+1
160 NEXT j
```

Usando las estructuras de las técnicas vistas anteriormente, resulta sencillo realizar rutinas para la creación de caracteres más altos o gruesos (caracteres con una altura el doble de lo normal, pero de anchura normal, o caracteres de altura normal y el doble de anchura). Una de las cosas que se suele hacer es prescindir de la mitad de la rutina que no se desee utilizar (de los desplazamientos de bits a través de los registros D y E o de las cargas de ambos registros D y E en la zona de memoria dedicada a los caracteres gráficos).

La principal diferencia entre los dos programas y el que duplica el tamaño de los caracteres está en la realización de los bucles, como es el caso de los que controlan la forma en que los bits se representan en los nuevos gráficos. Los tres programas utilizan la misma sección inicial para encontrar la dirección de los caracteres deseados en la zona de caracteres de la memoria. Yo he direccionado la zona de caracteres de una forma indirecta, así como la UDG, para que en el caso de que quieras usar un conjunto de caracteres que tú hayas creado, en una dirección diferente de la que usa el Sinclair para su conjunto de caracteres, puedas direccionarlo fácilmente.

En la rutina para hacer los caracteres más altos, no tenemos que realizar ningún desplazamiento de bits. Nosotros únicamente cargamos cada bit dos veces en posiciones adyacentes en la UDG, direccionada por el registro IX. Cuando un UDG se ha llenado, la rutina pasa automáticamente a la siguiente.

## LETRAS ALTAS

```
F000              ORG  #F000
F000 210000       LD   HL,#0000
F003 3A085C       LD   A,(#5C08)
F006 D620         SUB  #20
F008 6F           LD   L,A
F009 29           ADD  HL,HL
F00A 29           ADD  HL,HL
F00B 29           ADD  HL,HL
F00C ED4B365C     LD   BC,(#5C36)
F010 04           INC  B
F011 09           ADD  HL,BC
F012 DD2A7B5C     LD   IX,(#5C7B)
F016 0608         LD   B,#08
```

```
F018 7E          140       LD    A,(HL)
F019 DD7700      150       LD    (IX),A
F01C DD7701      160       LD    (IX+1),A
F01F 23          170       INC   HL
F020 DD23        180       INC   IX
F022 DD23        190       INC   IX
F024 10F2        200       DJNZ  #F018
F026 C9          210       RET
```

Para generar las letras anchas desplazamos (como en el caso de los caracteres duplicados), pero no los duplicamos.

## LETRAS ANCHAS

```
F000             10        ORG   #F000
F000 210000      20        LD     HL,0000
F003 3A085C      30        LD     A,(#5C08)
F006 D620        40        SUB   #20
F008 6F          50        LD     L,A
F009 29          60        ADD   HL,HL
F00A 29          70        ADD   HL,HL
F00B 29          80        ADD   HL,HL
F00C ED4B365C    90        LD     BC,(#5C36)
F010 04          100       INC   B
F011 09          110       ADD   HL,BC
F012 DD2A7B5C    120       LD     IX,(#5C7B)
F016 0E08        130       LD     C,#08
F018 7E          140       LD     A,(HL)
F019 0608        150       LD     B,#08
F01B 1F          160       RRA
F01C CB1A        170       RR     D
F01E CB1B        180       RR     E
F020 CB2A        190       SRA   D
F022 CB1B        200       RR     E
F024 10F5        210       DJNZ  #F01B
F026 DD7200      220       LD     (IX),D
F029 DD7308      230       LD     (IX+#08),E
F02C 23          240       INC   HL
F02D DD23        250       INC   IX
F02F 0D          260       DEC   C
F030 20E6        270       JR     NZ,#F018
F032 C9          280       RET
```

Como aplicación final de las rutinas que utilizan desplazamientos y rotaciones, he aquí un programa que te permitirá hacer impresiones con letras en negrita. En letras, se utiliza la denominación negrita para aquellas cuyos rasgos son más gruesos de lo normal, debido a lo cual resaltan más en el papel. En este programa conseguimos dicho efecto rotando cada byte de la letra y después realizando una función OR con el byte original (véase capítulo 6, programa para la introducción de caracteres de cuatro bits). Esto produce como efecto el duplicar los bits que estaban a uno en el byte original, haciendo más marcada cada línea de la letra. Para salir de la rutina de trabajar siempre en ensamblador, doy el programa en BASIC en una versión LPRINT.

LETRAS EN NEGRITA

```
F000                ORG   #F000
F000 210000         LD     HL,0000
F003 3A085C         LD     A,(#5C08)
F006 D620           SUB   #20
F008 6F             LD     L,A
F009 29             ADD   HL,HL
F00A 29             ADD   HL,HL
F00B 29             ADD   HL,HL
F00C ED4B365C       LD     BC,(#5C36)
F010 04             INC   B
F011 09             ADD   HL,BC
F012 ED5B7B5C       LD     DE,(#5C7B)
F016 0608           LD     B,#08
F018 7E             LD     A,(HL)
F019 1F             RRA
F01A B6             OR      (HL)
F01B 12             LD      (DE),A
F01C 23             INC   HL
F01D 13             INC   DE
F01E 10F8           DJNZ  #F018
F020 C9             RET
```

**Impresión en negrita**

```
100 PRINT "Impresion con";
110 LET W$="letras en negrita"
120 FOR j=1 TO LEN w$: POKE 23
560,CODE w$(j): RANDOMIZE USR 61
440: LPRINT " ";: NEXT j
```

La versión en negrita de cada letra se carga, de nuevo, en la tantas veces utilizada UDG "A".

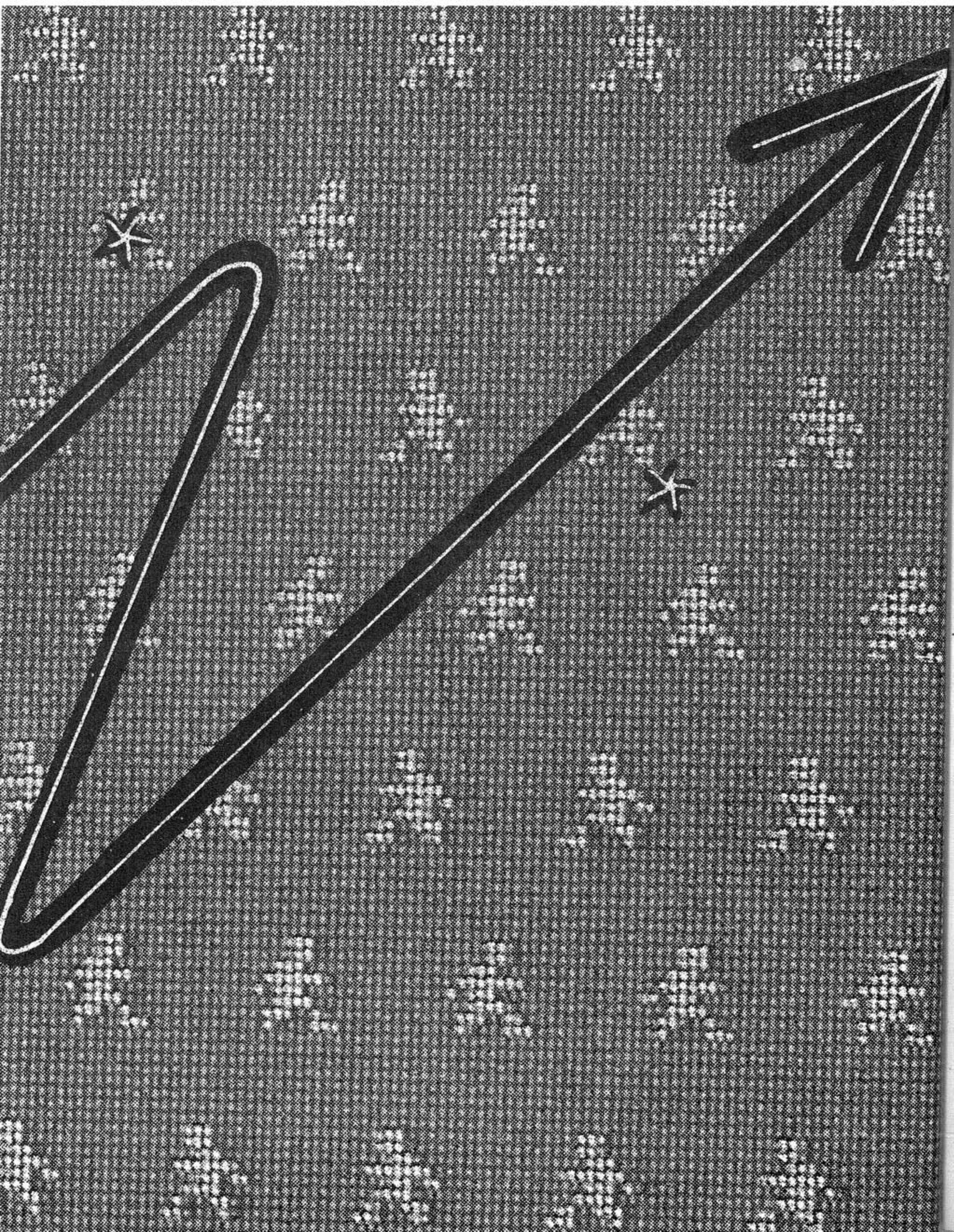
Y, por supuesto, hay más variaciones...

ESTA ES UNA LINEA EN NEGRITA

Esta rutina puede adaptarse de una manera muy sencilla para oscurecer toda la pantalla. El motivo por el que pudieras desear esto sería el de rellenar los "puntos claros" en una figura. A veces usas una rutina para gráficos a la que se supone que se va a rellenar pudiendo formar figuras compactas (generando, por ejemplo, una serie de curvas que completen la figura). A menudo esas curvas no se superponen completamente unas a otras, produciendo los antes mencionados "puntos claros". La rutina usada anteriormente para las letras en negrita se usará aquí para rellenar estos puntos. El resultado será:

## Pantalla en negrita

```
 10   ORG #F000
 20   LD  HL,#4000
 30   LD  BC,#1800
 40   LD  A,(HL)
 50   RRA
 60   OR  (HL)
 70   LD  (HL),A
 80   INC HL
 90   DEC BC
100   LD  A,B
110   OR  C
120   JR  NZ,#F006
130   RET
```



# 4
# Caracteres aún más grandes

Para la impresión del tamaño de las siguientes letras (4 veces mayores de lo normal, en vez del doble) se acude a una técnica un tanto diferente de la usada en el capítulo anterior.

Podríamos ampliar el sistema de rotaciones para, de una manera sencilla, producir cuatro caracteres gráficos por línea, en vez de dos. Sin embargo, Sinclair proporciona un conjunto completo de gráficos de 4 × 4 puntos de imagen. El único problema será, por tanto, acceder al gráfico que deseamos para cada pieza de nuestro gran carácter. Y para este problema, la forma en la que los gráficos de Sinclair están organizados ofrece una solución sencilla, como explicaré más adelante.

La figura 4.1 muestra la letra "B" impresa con una rejilla, la cual trocea a la letra en caracteres gráficos pertenecientes al conjunto definido en el Sinclair.

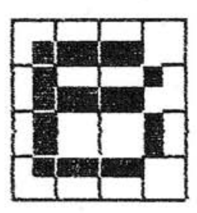
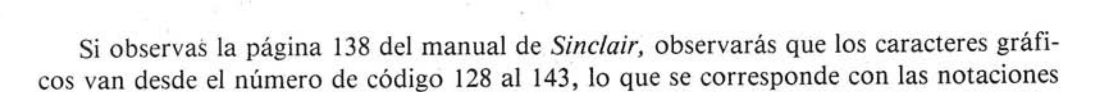


*Fig. 4.1*



Si observas la página 138 del manual de *Sinclair,* observarás que los caracteres gráficos van desde el número de código 128 al 143, lo que se corresponde con las notaciones

hexadecimales "80 + 0" y "80 + F". Ahora, si numeras las cuatro partes de cada carácter como se muestra en la figura 4.2, te darás cuenta de que el número a sumar a 80 en cada carácter gráfico se corresponde con el número de partes en negro de cada carácter gráfico.




*Fig. 4.2*

Por ejemplo, el carácter gráfico (correspondiente a la esquina superior izquierda de la letra "B" de la figura 4.1) tiene el código 132, que es el resultante de hacer 80h + 4, donde 4 es el número de la parte inferior derecha de nuestro cuadrado numerado. De igual forma se deduce que el siguiente carácter gráfico hacia la derecha, que forma la letra "B", tendrá como código 140, que resulta de hacer 80h + Ch, donde Ch equivale a 12 en decimal.

Resulta muy fácil calcular estos números para cada carácter gráfico. Por ejemplo, si extraes los "bits" 7 y 8 del primer byte que forma la letra "B" normal, tendrás como resultado "00". Haciendo lo mismo con el segundo byte de la misma letra, tendrás "01". Ordenando estos cuatro bits de la forma en que fueron extraídos (de derecha a izquierda y de abajo hacia arriba) tendrás la secuencia "0100", que como observarás tiene equivalente, en decimal, 4.

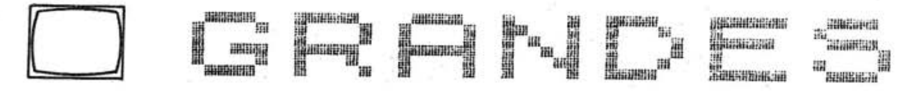
Ahora podemos incorporar una rutina en código que realice esto de una forma automática para cada bloque de cuatro bits.

La forma de hacerlo será cogiendo parejas de bytes procedentes del carácter y metiéndolas en los registros D y E; después efectuar dos operaciones de rotación, primero del registro E y después del registro D, pasando los cuatro bits deseados, en el orden correcto, al registro A. Después de esto sólo quedará por realizar la puesta a 1 del bit 7 del registro A (el cual da el valor 80h necesario en la suma) e imprimirlo.

Se da a continuación la rutina completa. De nuevo las instrucciones desde F000h hasta F011h tienen como objetivo el colocar las direcciones adecuadas en HL.

La parte de la rutina desde F02Ch hasta F03Bh lleva a cabo el movimiento de la posición de impresión, que consiste en bajarla una línea y desplazarla hacia la izquierda cuatro columnas, para que el próximo grupo de cuatro caracteres gráficos se imprima debajo del último. Como esto no lo has hecho antes, te indicaré cómo lo realiza la rutina. Sin embargo, habrás de conservar tu programa en BASIC para reasignar la posición de impresión de cada letra.

**Rutina para agrandar caracteres**



```
F000            10    ORG   #F000
F000 210000     20    LD     HL,#0000
F003 3A085C     30    LD     A,(#5C08)
F006 D620       40    SUB   #20
F008 6F         50    LD     L,A
F009 29         60    ADD   HL,HL
F00A 29         70    ADD   HL,HL
F00B 29         80    ADD   HL,HL
F00C ED4B365C   90    LD     BC,(#5C36)
F010 04        100    INC   B
F011 09        110    ADD   HL,BC
F012 0E04      120    LD     C,#04
F014 0604      130    LD     B,#04
F016 56        140    LD     D,(HL)
F017 23        150    INC   HL
F018 5E        160    LD     E,(HL)
F019 23        170    INC   HL
F01A AF        180    XOR   A
F01B CB13      190    RL     E
F01D 17        200    RLA
F01E CB13      210    RL     E
F020 17        220    RLA
F021 CB12      230    RL     D
F023 17        240    RLA
F024 CB12      250    RL     D
F026 17        260    RLA
F027 CBFF      270    SET   7,A
F029 D7        280    RST   #10
F02A 10EE      290    DJNZ  #F01A
F02C C5        300    PUSH  BC
F02D E5        310    PUSH  HL
F02E ED4B885C  320    LD     BC,(#5C88) ;Dir. variable S POSN
F032 05        330    DEC   B          ;Pasa a la linea sigiente
F033 0C        340    INC   C          ;Retrocede el cursor
F034 0C        350    INC   C          ; 4 posiciones para que
F035 0C        360    INC   C          ; la siguiente impresion
F036 0C        370    INC   C          ; se realice bajo la anterior
F037 CDD90D    380    CALL  #0DD9
F03A E1        390    POP   HL
F03B C1        400    POP   BC
F03C 0D        410    DEC   C
F03D 20D5      420    JR     NZ,#F014
F03F C9        430    RET
```

Para cambiar la posición de impresión, en primer lugar, has de encontrar la posición *actual.* La columna, así como la línea de impresión actuales, se encuentran almacenadas en la variable del sistema S_POSN. Esta variable contiene los números que usamos normalmente cuando imprimimos con "AT x,y", en el formato 33—X y 24—Y (donde X = columna e Y = línea). Pero para cambiar la posición de impresión no es suficiente con cambiar estos dos números; la variable del sistema DF_CC ha de ser cambiada al mismo tiempo. Esta última variable contiene la dirección del primer byte del carácter en el fichero de pantalla *(display file)* y, tal como ya veremos, éste no es un número muy fácil de calcular.

Sin embargo, una vez que nosotros tenemos los nuevos valores para la variable S_POSN, que sí son fáciles de calcular, si los colocamos en el par de registros BC y hacemos una llamada (CALL) a una rutina residente en la ROM, en la dirección ODD9h, esta rutina calculará el nuevo valor de la variable DF_CC y cargará todas las variables del sistema que sean necesarias.

Esto puede resultar muy útil cuando planifique operaciones de impresión usando rutinas en código máquina. Recuerda que en el registro B has de meter el resultado de 24 — línea; en el registro C el de 33 — columna y, después de esto, es cuando has de llamar a la rutina en la ROM con una instrucción "CALL 0DD9h". La mayoría de los registros resultarán modificados por esta rutina que llamas, por lo que es necesario que asegures los contenidos de los registros que necesites usar después introduciéndolos en el *stack* de cálculo con las instrucciones PUSH necesarias.

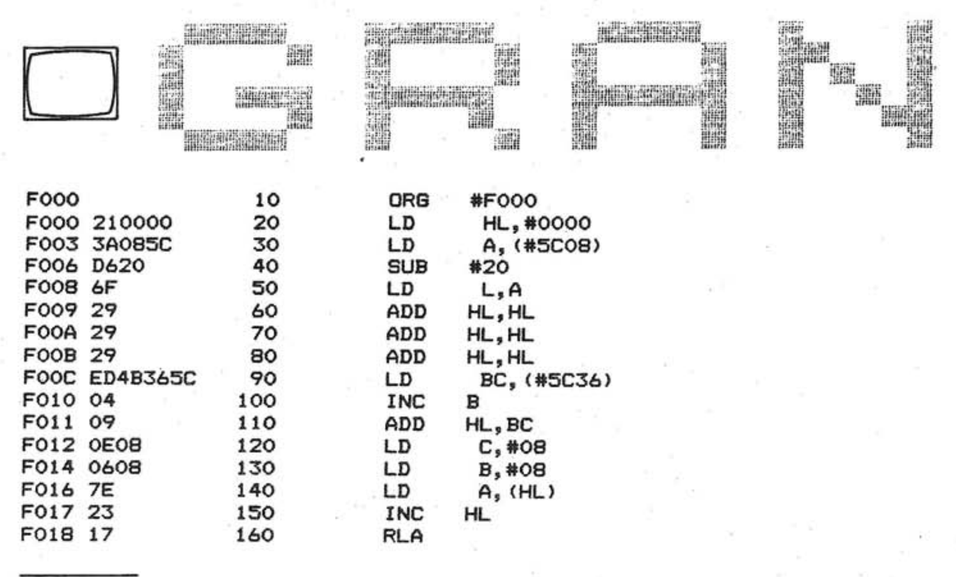
Lo último que quiero tratar sobre el tema de agrandar caracteres es el aumento en ocho veces de sus dimensiones longitudinales. Estas nuevas dimensiones requieren la utilización del espacio ocupado por el carácter original como una de las 64 partes iguales en que podemos dividir a este nuevo carácter, lo que da lugar a un carácter grande y con líneas más gruesas, pero que sólo usarás de vez en cuando, ya que sólo caben cuatro de estos caracteres en una línea.

Este tamaño de carácter es uno de los más fáciles de generar, ya que sólo hemos de observar los bytes del carácter ordenadamente, y, hecho esto, imprimiremos un cuadrado negro cuando encontremos un bit puesto a uno, y uno blanco cuando encontremos un cero.

Es evidente que hemos de usar el espacio gráfico, CHR$ 80h, en vez de espacio normal, CHR$ 20h. El cuadrado negro es CHR$ 8Fh; por tanto, sólo hemos de cambiar la segunda parte del código para obtener la impresión deseada: cuadrado blanco o negro. En la rutina dada a continuación esto se hace en las posiciones F017h a F01Fh; usando el bit de acarreo, que es cargado por instrucciones de rotación a la izquierda del registro A (RLA), para saltar sobre la operación no deseada.

Se utiliza la misma técnica para restablecer la posición de impresión, pero ahora hemos de mover esta posición *una* línea abajo y *ocho* posiciones a la izquierda.

**Rutina para generar caracteres de tamaño 8 × normal**




```
F000                  ORG  #F000
F000 210000           LD    HL,#0000
F003 3A085C           LD    A,(#5C08)
F006 D620             SUB  #20
F008 6F               LD    L,A
F009 29               ADD  HL,HL
F00A 29               ADD  HL,HL
F00B 29               ADD  HL,HL
F00C ED4B365C         LD    BC,(#5C36)
F010 04               INC  B
F011 09               ADD  HL,BC
F012 0E08             LD    C,#08
F014 0608             LD    B,#08
F016 7E               LD    A,(HL)
F017 23               INC  HL
F018 17               RLA
```

```
F019 F5          PUSH  AF
F01A 3E80        LD     A,#80    ;Codigo espacio en modo grafico
F01C 3002        JR     NC,#F020
F01E C60F        ADD    A,#0F    ;Pasa a espacio en negro
F020 D7          RST    #10      ;Imprime en pantalla
F021 F1          POP    AF
F022 10F4        DJNZ   #F018
F024 C5          PUSH   BC
F025 E5          PUSH   HL
F026 ED4B885C    LD     BC,(#5C88)
F02A 05          DEC    B        ;Pasa a linea siguiente
F02B 79          LD     A,C      ;Retrocede el cursor
F02C C608        ADD    A,08     ; ocho posiciones
F02E 4F          LD     C,A
F02F CDD90D      CALL   #0DD9
F032 E1          POP    HL
F033 C1          POP    BC
F034 0D          DEC    C
F035 20DD        JR     NZ,#F014
F037 C9          RET
```

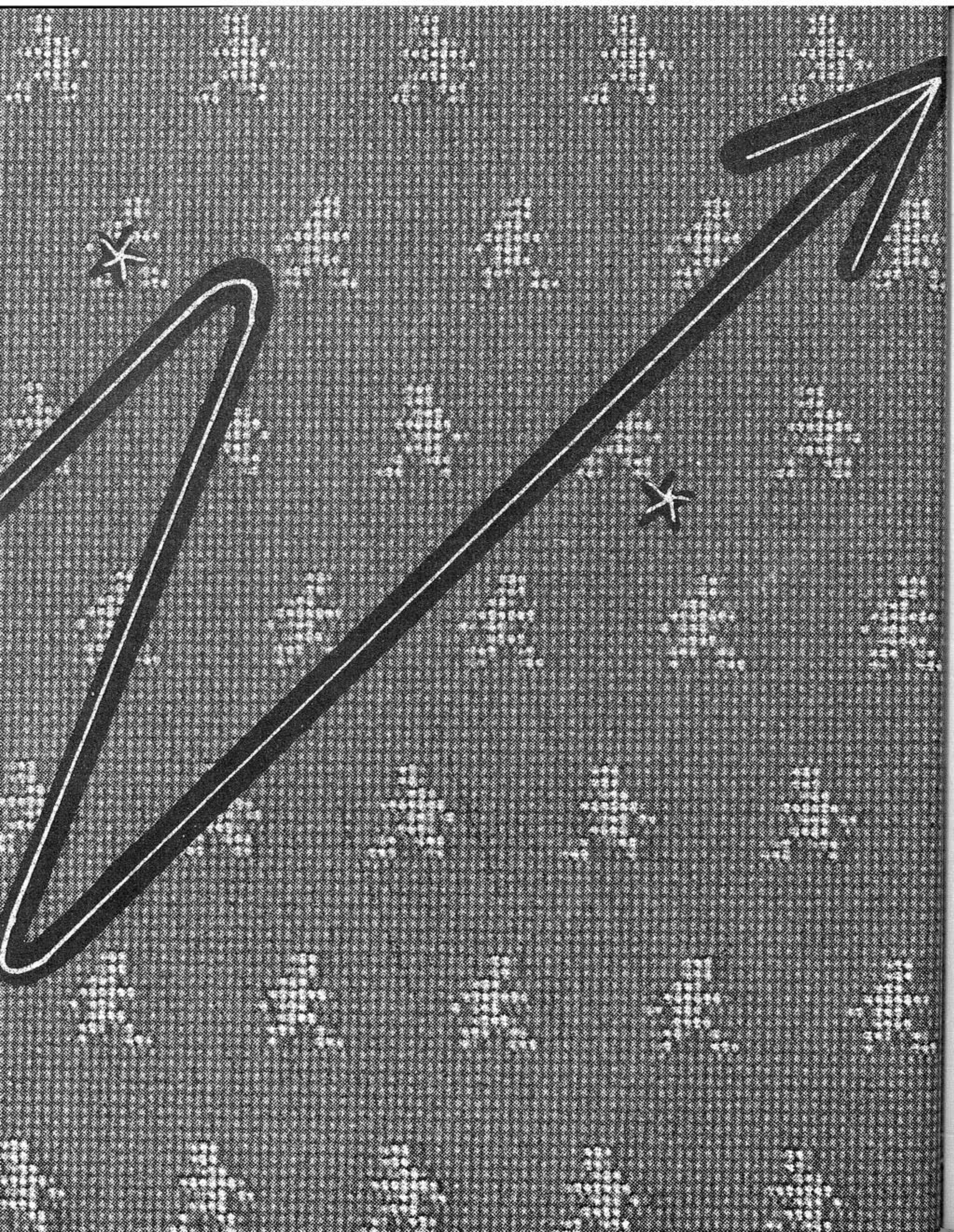
Debido a que ahora estamos trabajando con todas las posiciones de impresión para realizar la construcción de estos enormes caracteres, no hay ninguna razón por la que no podamos tener el mismo efecto usando el fichero de atributos, en vez del fichero de imagen, para contener los gráficos aumentados. Para este cambio no es necesaria una gran modificación de la rutina, únicamente poner la dirección del fichero de atributos en el par de registros DE y hacer un nuevo direccionamiento entre líneas usando una simple suma, en vez de la rutina de la ROM usada para el fichero de imagen.

**Rutina para generar caracteres de tamaño 8 × normal usando sólo el fichero de atributos**

```
F000             ORG    #F000
F000 210000      LD     HL,#0000
F003 111858      LD     DE,#5818
F006 3A085C      LD     A,(#5C08)
F009 D620        SUB    #20
F00B 6F          LD     L,A
F00C 29          ADD    HL,HL
F00D 29          ADD    HL,HL
F00E 29          ADD    HL,HL
F00F 01003D      LD     BC,#3D00
F012 09          ADD    HL,BC
F013 0E08        LD     C,08
F015 0608        LD     B,08
F017 7E          LD     A,(HL)
F018 23          INC    HL
F019 17          RLA
F01A F5          PUSH   AF
F01B 3E00        LD     A,#00    ;INK 0,PAPER 0
F01D 3802        JR     C,#F021
F01F 3E3F        LD     A,#3F    ;INK 7,PAPER 7
F021 12          LD     (DE),A
F022 13          INC    DE
F023 F1          POP    AF
F024 10F3        DJNZ   #F019
F026 C5          PUSH   BC
```

```
F027 EB              EX    DE,HL
F028 011800          LD    BC,#0018 ;Mueve (32-8)posiciones de impresion
F02B 09              ADD   HL,BC
F02C EB              EX    DE,HL   ;Lo introduce en DE
F02D C1              POP   BC
F02E 0D              DEC   C
F02F 20E4            JR    NZ,#F015
F031 C9              RET
```

El hecho de que no usemos el fichero de imagen, aunque los resultados aparezcan como impresiones en pantalla, nos genera una serie de curiosas e interesantes posibilidades. Veremos más usos de los ficheros de atributos e imagen en un posterior capítulo.



# 5

# Caracteres escritos en otros sentidos y direcciones distintos al normal

## Nuevo conjunto de caracteres

Hay varias posibilidades de impresión en el Spectrum que vamos a intentar utilizar en este capítulo. Hasta ahora sólo hemos tenido en cuenta las técnicas que producen caracteres en la forma usual, tal y como la pantalla los necesita para realizar una impresión. Esta técnica es adecuada cuando los nuevos rótulos son necesarios sólo en muy determinados momentos, siendo en este caso las rutinas que los crean bastante lentas en su ejecución, al crear un carácter cada vez; esta lentitud podría ser una desventaja para ti al querer producir rótulos que ocupen una gran parte de la pantalla.

Los nuevos rótulos, de los que hablaremos en este capítulo, se adaptan mejor a la utilización de un conjunto de caracteres totalmente nuevo. Estos pueden usarse cuando lo desees, cargando la variable del sistema CHARS mediante instrucciones POKE con la nueva dirección *menos* 100h (256d). El conjunto de caracteres residente en la ROM puede volver a ser utilizado almacenando en la misma variable anterior, CHARS, su dirección normal, 3C00h. El alfabeto actual comienza en la dirección 3D00h.

Una característica común entre todos estos nuevos alfabetos de caracteres es que se basan en el alfabeto existente de origen en el Spectrum. Pero no estoy queriendo decir con esto que tengas que teclear de nuevo 96 nuevos caracteres o más, de ocho bytes cada uno. El objetivo de este capítulo es enseñarte cómo escribir programas que generarán nuevos alfabetos de caracteres sobre la base del original del Spectrum.

## Caracteres en varias direcciones

El primer grupo modificado de caracteres que voy a tener en cuenta es el que utiliza las letras normales del Spectrum, pero imprimiéndolas en otras direcciones.

Esto puede ser muy útil para ti si quieres presentar resultados en gráficos sobre unos propósitos científicos y financieros. Esto se convierte en una obligación si tienes una pantalla cerrada en sentido horizontal (es decir, como si su parte derecha y la izquierda estuvieran unidas formando un cilindro), en la que has de colocar un título en una posición de dicha pantalla. La figura 5.1 muestra lo que quiero decir.

De hecho, los caracteres impresos en distintas direcciones *pueden* ser generados de uno en uno, como lo hemos venido haciendo hasta ahora. Pero si tienes espacio libre en la RAM para colocar un conjunto completo de caracteres (ocupan 300h [768d] bytes), esto es mucho más rápido y fácil para trabajar.

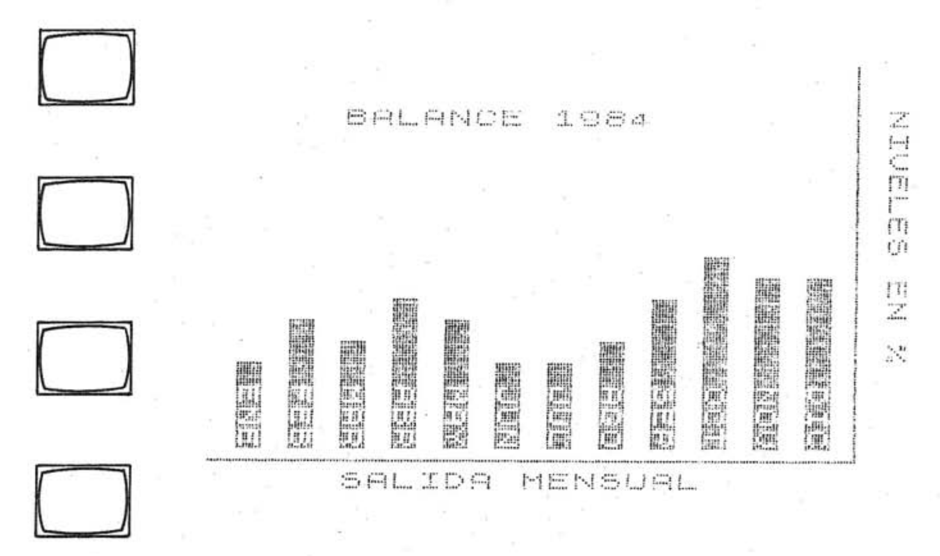



*Fig. 5.1. Conjunto de caracteres en distintas direcciones*

En primer lugar, hemos de ver lo que tenemos que hacer y cómo vamos a hacerlo.

## Giro de las letras sobre sus caras

Como sabemos, un carácter está compuesto de ocho bytes, cada uno de los cuales codifica una línea de impresión del carácter. La "A" mayúscula toma la apariencia de la figura 5.2.

Para colocar la "A" sobre cada una de sus caras, hemos de sacar los bits puestos a uno, de cada byte, uno cada vez, y reordenarlos en otros nuevos bytes, tomando la apariencia que se muestra en la figura 5.3.

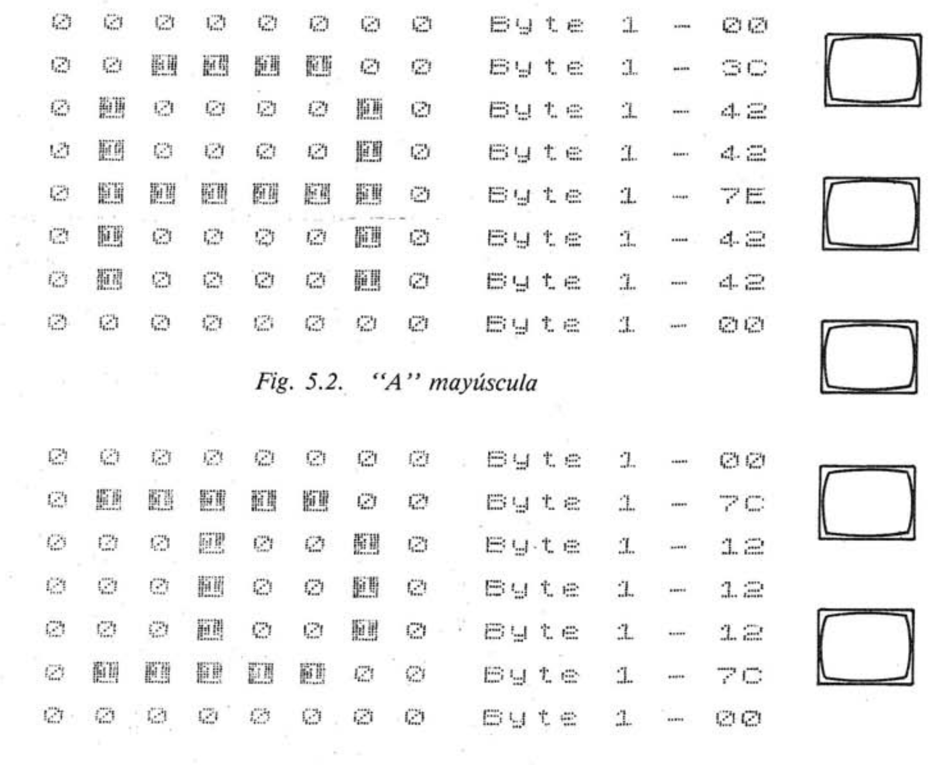



*Fig. 5.2. "A" mayúscula*



*Fig. 5.3. "A" mayúscula sobre su parte derecha*

Puede ver que el primer byte del nuevo carácter es una línea de ceros correspondiente a la columna izquierda de bits de la figura 5.2. El segundo byte, en la figura 5.3, se corresponde con la segunda columna de bits de la figura 5.2, y así sucesivamente.

Esto quiere decir, en términos de programación, que hemos de rotar cada byte original, cíclicamente, para tener cada vez un bit de dicho byte en el bit de acarreo. Cada vez que hacemos esto, sacamos el bit de acarreo y lo pasamos al nuevo byte que estamos construyendo.

Para realizar esas operaciones hemos de tener una zona dedicada a la creación de estos nuevos caracteres, ya que es imposible realizar rotaciones o cualquier otra operación en la ROM. Por tanto, lo primero que hemos de hacer será trasladar todos los grupos de ocho bytes del carácter original a una nueva dirección (yo te sugiero la que indica la variable del sistema "MEM", que contiene la dirección de la zona de memoria usada por el calculador del Spectrum, la cual es tan manejable como cualquier otra). Suponiendo que la dirección de nuestro carácter situado en "MEM" está en el par de registros HL, podemos realizar una "RL (HL)", seguida de un "INC HL" y por una "RRA" (o lo que es lo mismo: primero desplazamos hacia la izquierda el contenido de la posición de memoria indicado en "HL", situando su bit más significativo en el bit de acarreo; luego incrementamos en una unidad el contenido de "HL" para que apunte a la siguiente posición de memoria y, por último, pasamos el bit de acarreo al acumulador por la izquierda, es decir, poniéndolo como bit de mayor peso del acumulador); este pro-

ceso se repetirá siete veces más. Al finalizar este proceso tendremos el nuevo byte en el registro A.

A continuación se da el listado de dicho proceso, con HL conteniendo la dirección del carácter en la ROM:

### Letras en una sola dirección

```
F010             10      ORG    #F010
F010 11925C      20      LD      DE,#5C92
F013 D5          30      PUSH   DE
F014 010800      40      LD      BC,#0008
F017 EDB0        50      LDIR          ;Mete caracter en zona para giros
F019 E1          60      POP    HL      ;Mete en HL Dir. zona giros
F01A 1158FF      70      LD      DE,#FF58 ;Apunta a UDG "A"
F01D 0E08        80      LD      C,08
F01F 0608        90      LD      B,08
F021 E5         100      PUSH   HL
F022 CB16       110      RL      (HL)
F024 23         120      INC    HL
F025 1F         130      RRA
F026 10FA       140      DJNZ   #F022
F028 12         150      LD      (DE),A  ;Carga nuevo byte en UDG
F029 13         160      INC    DE
F02A E1         170      POP    HL      ;Recupera Dir. zona giros
F02B 0D         180      DEC    C
F02C 20F1       190      JR      NZ,#F01F
F02E C9         200      RET
```

Para poder realizar la transformación en la forma que hemos pensado, hay que añadir la cabecera estándar al programa, la cual recupera el código del carácter contenido en la variable del sistema LAST_K.

### Impresión en varias direcciones

```
F000             10      ORG    #F000
F000 210000      20      LD      HL,0000
F003 3A085C      30      LD      A,(#5C08)
F006 D620        40      SUB    #20
F008 6F          50      LD      L,A
F009 29          60      ADD    HL,HL
F00A 29          70      ADD    HL,HL
F00B 29          80      ADD    HL,HL
F00C 01003D      90      LD      BC,#3D00
F00F 09         100      ADD    HL,BC
F010 11925C     110      LD      DE,#5C92
F013 D5         120      PUSH   DE
F014 010800     130      LD      BC,#0008
F017 EDB0       140      LDIR
F019 E1         150      POP    HL
F01A 1158FF     160      LD      DE,#FF58
F01D 0E08       170      LD      C,08
F01F 0608       180      LD      B,08
F021 E5         190      PUSH   HL
F022 CB16       200      RL      (HL)
F024 23         210      INC    HL
```

```
F025 1F          RRA
F026 10FA        DJNZ  #F022
F028 12          LD     (DE),A
F029 13          INC   DE
F02A E1          POP   HL
F02B 0D          DEC   C
F02C 20F1        JR     NZ,#F01F
F02E C9          RET
```

Para generar el alfabeto completo necesitas una dirección de comienzo para el nuevo grupo de caracteres; también hacer algunas modificaciones a la rutina.

Es aconsejable que utilices una dirección de comienzo para el nuevo alfabeto cuyos dos últimos dígitos sean "00".

Esto se debe a que el alfabeto residente en la ROM comienza en la dirección 3D00h, por lo que, si la nueva dirección también acabase en 00, sólo sería necesario que cambiases el primer byte para que puedas utilizar tu nueva dirección en la misma rutina. Esto significa que has de cambiar el número contenido en la dirección "5C37h" (CHARS + 1, 23607d) para cambiar de alfabeto. Te sugiero que utilices la dirección "E000h".

La rutina utiliza registros alternativos para guardar el número total de caracteres del alfabeto (60h o 96d) y para manejar la transferencia desde la ROM a la zona de creación de los nuevos caracteres. Observa en el listado de la rutina cómo lo primero que realiza es un almacenamiento en el *stack* (PUSH HL), por lo que la última acción antes de salir de la rutina es la recuperación de HL del *stack* (POP HL).

Esto permite mantener intacta la dirección que había en dicho par de registros antes de llamar a la rutina. Observa también cómo después de cada transferencia desde la ROM a la zona de generación de los caracteres, el par de registros HL estarán apuntando a la próxima letra, ya que su contenido ha sido incrementado adecuadamente mediante la instrucción "LDIR".

### Alfabeto de caracteres en distintas orientaciones

```
F000             ORG   #F000
F000 1100E0      LD     DE,#E000 ;Dir. comienzo nuevo alfabeto
F003 D9          EXX
F004 E5          PUSH  HL
F005 0660        LD     B,#60
F007 21003D      LD     HL,#3D00 ;Dir. comienzo alfabeto en ROM
F00A C5          PUSH  BC
F00B 11925C      LD     DE,#5C92 ;Zona de giros
F00E 010800      LD     BC,#0008
F011 EDB0        LDIR
F013 D9          EXX
F014 21925C      LD     HL,#5C92
F017 0E08        LD     C,08
F019 0608        LD     B,08
F01B E5          PUSH  HL
F01C CB16        RL     (HL)
F01E 23          INC   HL
F01F 1F          RRA
F020 10FA        DJNZ  #F01C
F022 12          LD     (DE),A
F023 13          INC   DE
```

```
F024 E1     POP   HL
F025 0D     DEC   C
F026 20F1   JR     NZ,#F019
F028 D9     EXX
F029 C1     POP   BC
F02A 10DE   DJNZ  #F00A  ;Salta a siguiente caracter
F02C E1     POP   HL
F02D D9     EXX
F02E C9     RET
```

Después de introducir y ejecutar la rutina, crearás un alfabeto de caracteres totalmente nuevo a partir de la dirección E000h. Si escribes un programa en BASIC al que añades la línea "POKE 23607,223", todo lo que imprimas serán caracteres girados hacia la derecha.

Pero recuerda que has de añadir la instrucción "POKE 23607,60" en algún lugar, pues puede que te sorprendas con alguno de los posibles caracteres extraños que se impriman.

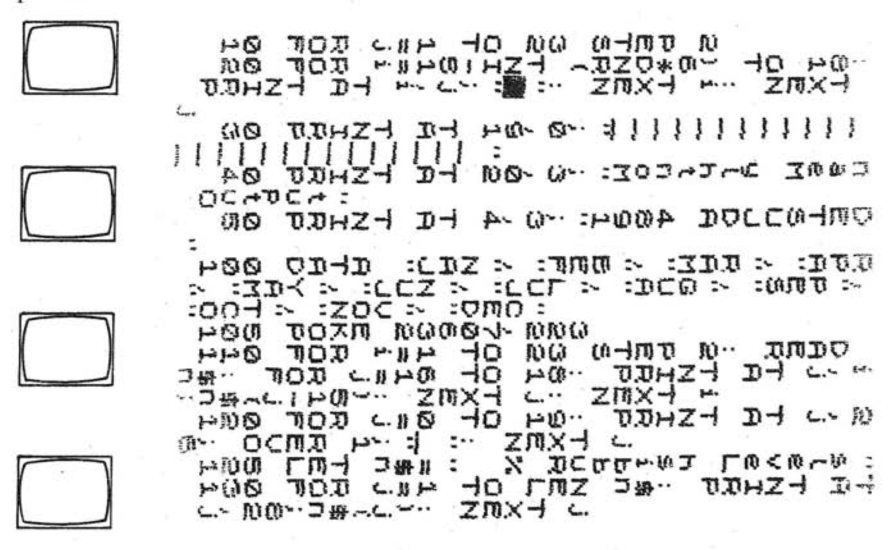


*Fig. 5.4*

Un último apunte sobre este tema: las operaciones clave, como habrás podido observar a lo largo del capítulo, son "RL (HL)" y "RRA". Si cambias alguna de estas instrucciones, alteras la secuencia en la que se van formando los bytes de los nuevos caracteres, por lo que puedes crear caracteres impresos en otras direcciones.

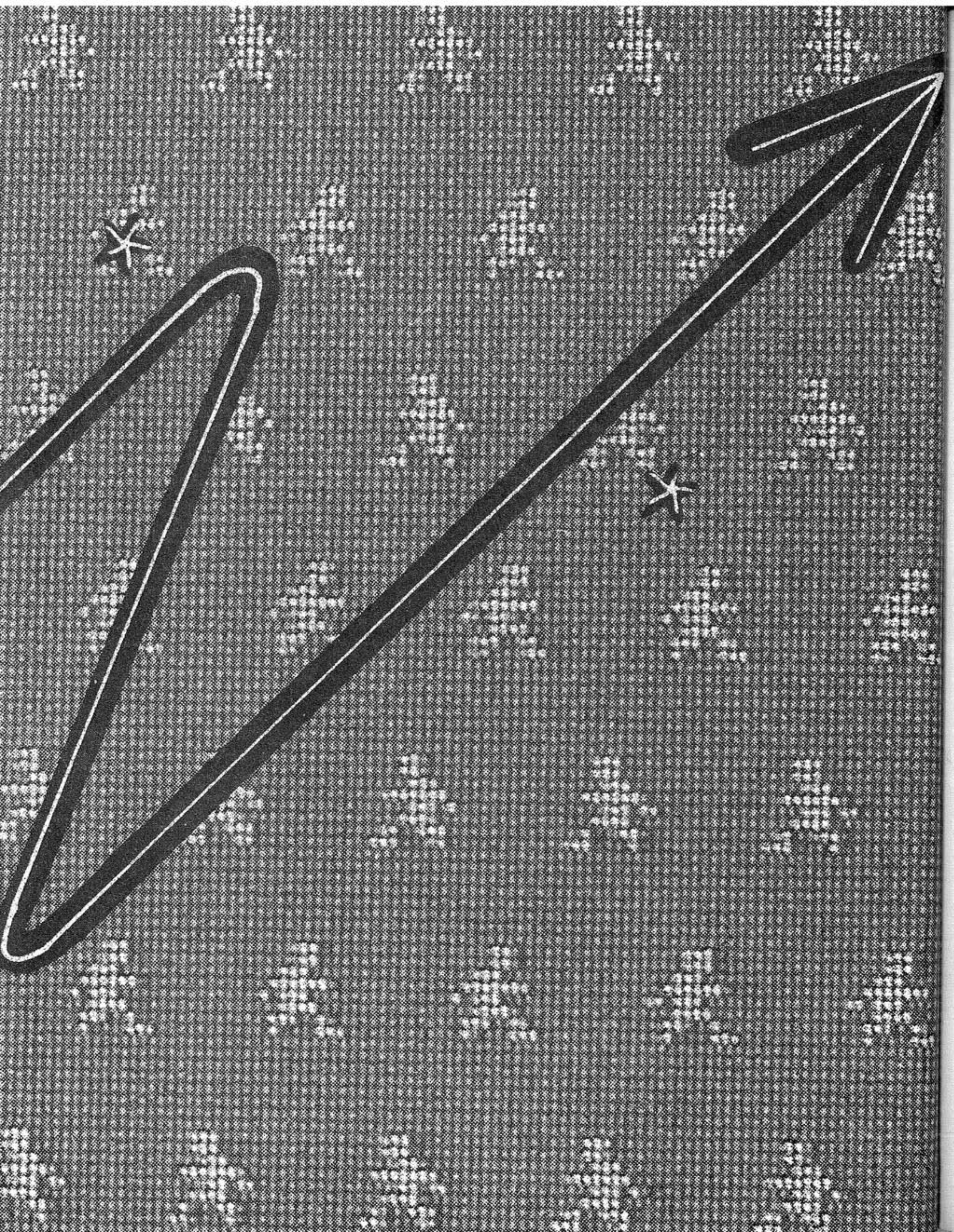
RL (HL) + RRA

RR (HL) + RRA

RL (HL) + RLA

RR (HL) + RLA

*Fig. 5.5. Caracteres impresos en varias direcciones*



# 6
# Caracteres pequeños

Como he indicado anteriormente, el alfabeto del Spectrum utiliza más bits para la creación de sus caracteres que los que son absolutamente necesarios para producir un conjunto de caracteres perfectamente legibles. Reduciendo el número de bits usados en la impresión horizontal de los caracteres, es decir, estrechando los caracteres, teóricamente podrías imprimir más caracteres por línea.

Una de las cosas que nos gustaría poder hacer sería introducir trozos enteros de texto definibles en tamaño e incluso poder definir su tamaño, aunque el texto procediese de la memoria del Spectrum. Con 32 caracteres por línea no podemos introducir demasiadas cosas en la pantalla, por lo que el efecto es más de estar leyendo un telegrama que un texto.

## Caracteres de seis bits

Esta es una versión del alfabeto del Spectrum bastante sencilla de crear. Tú puedes generar el nuevo alfabeto escogiendo solamente seis de los ocho posibles bits que forman cada línea de un carácter normal (figura 6.1).

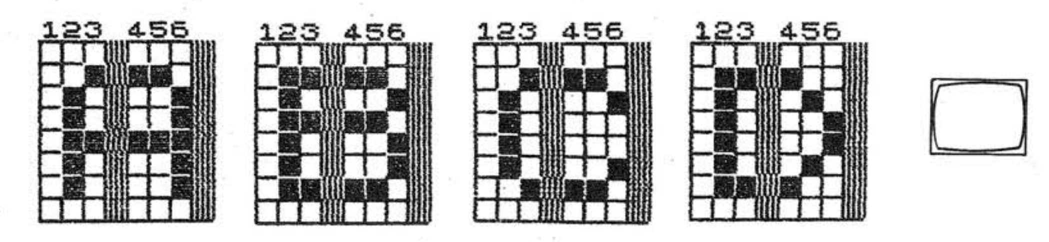



*Fig. 6.1*

El último bit de todas las filas, en la versión normal del Spectrum, está siempre a cero, por lo que tú solamente estás quitando un bit significativo por byte. La figura 6.2 muestra un alfabeto completo de letras mayúsculas generadas de esta manera. Como puedes comprobar, su aspecto es bastante convincente, con la excepción de la T y la Y, que han sufrido serias mutilaciones, y de la I, que ha quedado asimétrica.

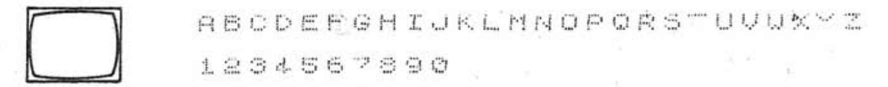


*Fig. 6.2*

Las instrucciones básicas en código máquina que permiten la obtención de estos caracteres vuelven a ser las rotaciones y los desplazamientos. La rutina carga cada byte del carácter normal en el registro A y después carga, uno a uno, los bits que lo forman, en el bit de acarreo. Una vez el bit en el bit de acarreo, la rutina lo toma si lo necesita, despreciándolo en caso contrario, para formar el nuevo carácter y los transfiere a un nuevo byte residente en una nueva dirección que es apuntada por el par de registros HL.

## Caracteres de seis bits

```
F000               ORG    #F000
F000 11003D        LD      DE,#3D00
F003 2100E0        LD      HL,#E000
F006 010003        LD      BC,#0300
F009 1A            LD      A,(DE)
F00A 07            RLCA
F00B CB16          RL      (HL)
F00D 07            RLCA
F00E CB16          RL      (HL)
F010 07            RLCA
F011 CB16          RL      (HL)
F013 07            RLCA
F014 07            RLCA
F015 CB16          RL      (HL)
F017 07            RLCA
F018 CB16          RL      (HL)
F01A 07            RLCA
F01B CB16          RL      (HL)
F01D CB26          SLA    (HL)
F01F CB26          SLA    (HL)
F021 EDA0          LDI
F023 E0            RET    PO
F024 18E3          JR      #F009
```

La instrucción de carga automática "LDI" es muy útil para esta rutina, así como la instrucción de incrementar los pares de registros HL y DE y la de decrementar el contador residente en el par "BC".

Mientras que las letras mayúsculas e incluso los números pueden ser transformados usando esta técnica, las letras minúsculas no quedan con un aspecto muy agradable (figura 6.3): la "a" y la "b" quedan bien, pero la "t" y la "f" quedan bastante mal.

Yo exageraba un poco al decir que todos estos nuevos caracteres de seis bits podían ser generados automáticamente a partir del alfabeto original del Sinclair. Muchos de ellos sí pueden generarse de esta manera, pero siempre habrá excepciones que tendrán que ser ajustadas individualmente.

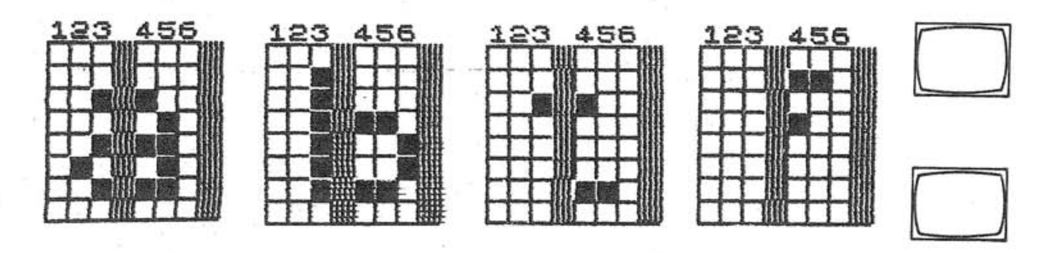



*Fig. 6.3*

Para realizar el ajuste individual he creado un programa, llamado Titivator, que te permitirá retocar cualquier carácter que desees, una vez que hayas generado una versión aproximada que esté almacenada a partir de la dirección E000h. El programa te dibujará en la pantalla versiones aumentadas de cualquiera de los nuevos caracteres que hayas creado y te permitirá introducir el código binario de cada línea para construir un carácter ya acabado. La figura 6.4 muestra su apariencia en la pantalla al ejecutar este programa.

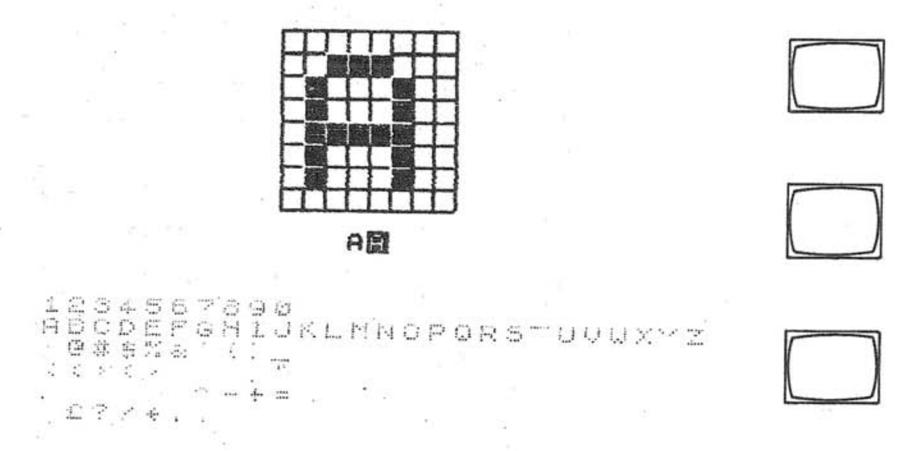


*Fig. 6.4*

Con la sencilla operación de pulsar una de las teclas del Spectrum, aparecerá en la pantalla una versión agrandada del carácter que has elegido y otra versión en tamaño normal (con fondo normal y en negativo), así como el alfabeto completo debajo de ambos dibujos. Si quieres modificar algún carácter, lo puedes hacer pulsando la tecla ENTER cada vez que quieras introducir una línea, es decir, un byte, en binario. No es

necesario que introduzcas, cada vez que realizas esta operación, todos los ceros finales, ya que la secuencia "0101" es igual a la "01010000". Antes de escribir el programa debes introducir el código de la rutina para generar caracteres en tamaño ocho veces superior al normal, visto al final del capítulo 4, en una declaración REM, en la línea 1 (esto te ocupará un equivalente a 54 espacios). Hecho esto, introduce el resto del programa en BASIC.

**Titivator**

```
   5 POKE 23660,5
   6 POKE USR "a",255: FOR j=1 T
O 7: POKE USR "a"+j,128: NEXT j
   7 POKE USR "b",255: FOR j=1 T
O 7: POKE USR "b"+j,0: NEXT j
   8 FOR j=0 TO 7: POKE USR "c"+
j,128: NEXT j
   9 INPUT "Introduce dir. comie
nzo nuevo alfabeto";ad: LET ad=a
d-256: LET ad2=INT (ad/256): LET
 ad1=ad-256*ad2
  10 PRINT #1;AT 0,0;"Caracter?"
: PAUSE 0: LET y$=INKEY$: PRINT
#2
  20 IF CODE y$=13 THEN  GO TO 2
00
  25 LET chr=CODE y$
  30 POKE 23606,ad1: POKE 23607,
ad2: POKE 23560,chr
  50 LET z$="AAAAAAAC"
  60 FOR j=0 TO 7: PRINT AT 2+j,
11;z$: NEXT j
  70 PRINT  AT 10,11;"BBBBBBBB"
  80 PRINT  OVER 1;AT 2,11;: RAN
DOMIZE USR (5+PEEK 23635+256*PEE
K 23636)
  90 PRINT AT 11,14;CHR$ chr; IN
VERSE 1;CHR$ chr
 100 PRINT AT 18,0;: FOR j=32 TO
 127: PRINT CHR$ j;: NEXT j
 110 POKE 23606,0: POKE 23607,60
 120 GO TO 10
 200 DIM b$(8)
 210 FOR j=0 TO 7
```

```
220 INPUT "Linea numero:";(j+1)
;"BIN";b$
230 IF b$(1)<>"0" AND b$(1)<>"1
" THEN STOP
240 LET x=0: FOR i=1 TO 8: LET
x=x*2+(b$(i)="1"): NEXT j
250 POKE ad+256+8*(chr-32)+j,x
260 NEXT j
270 GO TO 30
```

No creo que el programa en BASIC sea complicado de entender, pero por si acaso voy a darte unas pequeñas referencias de cómo trabaja:

— *Línea 5:* Introduciendo este valor en la posición 23660, que es la variable del sistema "S_TOP", el listado automático comenzará a partir de dicha línea 5; con esto se evita que aparezca listado el programa en código máquina, ya que esto haría más complicada la lectura y comprensión del resto del programa.

— *Líneas 6 a 8:* Aquí se generan los gráficos redefinibles por el usuario, que crean la rejilla que aparece en el carácter aumentado.

— *Línea 10:* La línea 10 hace un cambio de canal para, de esta forma, imprimir en la parte baja de la pantalla y después lo cambia de nuevo.

— *Línea 30:* Cambia al nuevo alfabeto de caracteres (en la dirección 57344d) y coloca el código del carácter seleccionado en la variable del sistema "LAST_K".

— *Líneas 50 a 70:* Imprime la rejilla.

— *Línea 80:* Comienza a ejecutar el programa en código máquina. La dirección del programa en BASIC, situada en la variable del sistema PROG, puede cambiar, aunque en la práctica siempre permanece igual, a no ser que la "Interface" (adaptador de conexión) 1 esté conectada. Sin la "Interface" puedes hacer lo mismo que se hace en la línea 80, usando tan sólo la instrucción "RANDOMIZE USR 23760", pero es más seguro el uso del direccionamiento indirecto.

— *Línea 110:* Cambia de nuevo a los caracteres normales.

La figura 6.5 muestra el aspecto que toman los caracteres de seis bits después de utilizar el Titivator. Están, como podrás ver, muy mejorados. Ahora dejo que trabajes en la realización de los signos de puntuación, etc., en caracteres de seis bits.

ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
0123456789

*Fig. 6.5. Alfabeto de caracteres de seis bits*

Tal como está, el nuevo alfabeto no presenta ninguna ventaja sobre el viejo, ya que cada letra sigue ocupando una posición de impresión de 8 × 8 bits. Por tanto, para obtener una mejora, hemos de reducir el espacio entre las letras. Esto se puede hacer, pero se necesitan algunos programas en código máquina que pueden resultar complicados, motivo por el cual lo dejo para el próximo capítulo. Mientras tanto, podemos ver la forma de generar otro nuevo alfabeto de caracteres, más manejable que el de seis bits: el alfabeto de cuatro bits.

## Caracteres de cuatro bits

Aunque parezca sorprendente, puedes generar letras usando solamente una anchura de carácter de *tres bits,* más un bit extra para crear el espacio entre las letras. Puede que éstas no sean las letras más bonitas del mundo, pero podrás leerlas y podrás tener una línea con 64 letras en tu Sinclair; esto es equivalente, por tanto, a una hoja normal de impresión. Resulta atractiva esta nueva posibilidad de impresión, ya que permite colocar dos caracteres de cuatro bits en el lugar que ocupa un solo carácter de ocho bits. Además, este doble carácter se puede generar de una forma tan sencilla como es el uso de operaciones lógicas AND y OR, tal y como veremos más adelante.

El listado del programa generador en código máquina es casi el mismo que el que utilizaste para los caracteres de seis bits. Sin embargo, como ahora tenemos solamente tres bits para utilizar, hemos de hacer una selección más cuidadosa de estos tres bits. De hecho, creo que es mejor dividirlo en dos partes —la primera, selección para conseguir las letras minúsculas y, la segunda, para las letras mayúsculas—. Para el resto de los caracteres dejo a tu elección el método de realización.

El listado que sigue es el que, según mi opinión, es mejor para el caso de las minúsculas.

**Caracteres de cuatro bits**

```
F000              10      ORG    #F000
F000 11003D       20      LD      DE,#3D00
F003 2100FC       30      LD      HL,#FC00
F006 010003       40      LD      BC,#0300
F009 AF           50      XOR    A
F00A 77           60      LD      (HL),A
F00B 1A           70      LD      A,(DE)
F00C 07           80      RLCA
F00D CB16         90      RL      (HL)
F00F 07          100      RLCA
F010 CB16        110      RL      (HL)
F012 07          120      RLCA
F013 07          130      RLCA
F014 CB16        140      RL      (HL)
F016 07          150      RLCA
F017 07          160      RLCA
F018 CB16        170      RL      (HL)
F01A 07          180      RLCA
F01B EDA0        190      LDI
F01D E0          200      RET    PO
F01E 18E9        210      JR      #F009
```

Hay dos cosas que es importante que observes en este listado. La primera es que coloco el nuevo alfabeto en la dirección FC00h; como usarás 300h bytes de memoria para dicho alfabeto, esta posición resulta lo suficientemente alta en la RAM como para que puedas guardarlos en memoria sin tener que modificar los valores de UDG. Querrás guardar el alfabeto después de haberlo construido, conjuntamente con el programa que lo generó, y esto es mejor hacerlo guardando la parte de los datos en la zona alta de la RAM.

La segunda cuestión es que he incorporado dos instrucciones, en F009h y F00Ah, que ponen a cero el byte de memoria que apunta el par de registros HL. Si no haces esto, te aparecerán en los caracteres partes extrañas, según lo que hubiera antes en dicho byte.

La figura 6.6 muestra la apariencia que toma el alfabeto después de la primera operación (generación del alfabeto), si ya tienes en memoria el programa Titivator, con su línea 30 de la forma "POKE 23607,251...".

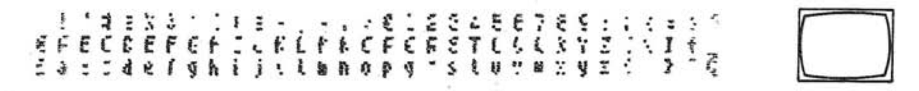


*Fig. 6.6*

Para crear las mayúsculas y los números has de cambiar las instrucciones de las líneas F018h y F01Ah, de forma que tengan la secuencia "RLCA RL (HL)", en vez de la que hay ahora de "RL (HL) RLCA". Habrás de cambiar también la línea F006h a "LD BC,0200h".

Después de ejecutar la rutina Titivator, el alfabeto de la figura 6.6 tomará el aspecto que aparece en la figura 6.7. Como puedes ver, ambas son unas versiones muy poco precisas, pero esta última comienza a ser reconocible y legible.

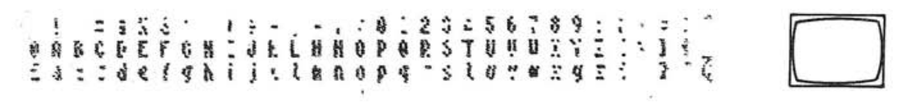


*Fig. 6.7*

Después de una última sesión con el Titivator, conseguirás obtener un alfabeto con el aspecto del de la figura 6.8. Como verás, éste no es del todo perfecto. Las letras que presentan una mayor dificultad para ser leídas son: "H", "M", "N", "W", "n" y "m". No hay suficiente información con tres bits para poder diferenciarlas de una forma clara. Tendrás que confiar en que lo que los ojos de la gente verán será lo que quieran ver, y esto se cumple con la mayoría de la gente.

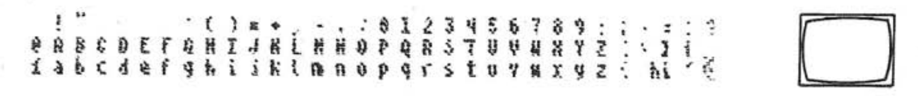


*Fig. 6.8*

A continuación te muestro las soluciones que he encontrado para las letras que dan problemas (figura 6.9); en la versión aumentada que viene en la figura 6.9, la solución no les da un aspecto muy convincente, pero quedan bien cuando están en un texto. Puede que consideres que alguna de estas variaciones harán que determinada letra mejore.

Una vez realizado y almacenado de forma segura en una cassette tu alfabeto de cuatro bits, has de pensar cómo lo vas a usar.

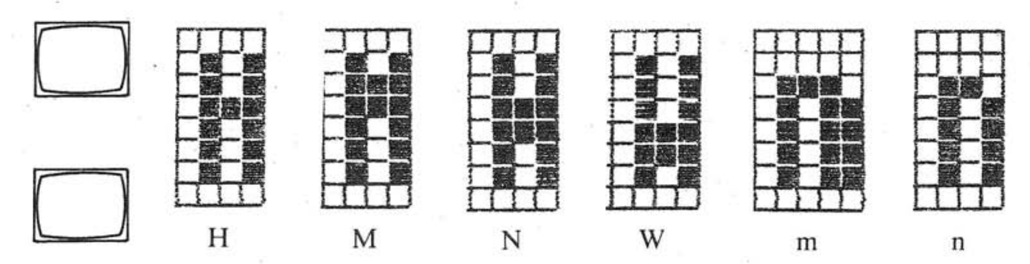



*Fig. 6.9. Soluciones para las letras con problemas*

Hay varias opciones. Puede que quieras usar este alfabeto de cuatro bits para imprimir textos o datos que tienes almacenados en la RAM. Puede que desees usar las letras como parte de un sistema de entrada de textos a la memoria RAM. Puede que, incluso, quieras mezclar las dos opciones para tener un procesador de textos.

Nosotros sólo consideraremos aquí la primera opción: la de imprimir textos. He dado el nombre de "Máquina de escribir" a esta opción.

## Máquina de escribir

Para realizar esto, hemos de sacar los caracteres del teclado del Spectrum, como hemos hecho anteriormente, buscando, en primer lugar, su dirección en el alfabeto.

Hecho esto, aparece aquí una diferencia con respecto a lo que hacíamos anteriormente. Hemos de encontrar la forma de mostrar un espacio de 8 × 8 bits compuesto de dos pulsaciones de teclas consecutivamente. Para hacerlo, lo más sencillo es dividir el programa en dos partes. Una subrutina llamada desde el programa principal, que encuentra el carácter (y busca la forma, después de hacer otra serie de cosas, de salirse de la secuencia normal e imprimir usando secuencias de borrando y nueva línea). La rutina principal combina los caracteres y los imprime.

La subrutina utiliza elementos que nos son ya familiares, por lo que vamos a ver cómo funciona.

### Caracteres de cuatro bits. Subrutina

```
F130              10        ORG    #F130
F130 FDCB01AE     20        RES    5,   (IY+01)
F134 FDCB016E     30        BIT    5,   (IY+01)
F138 28FA         40        JR      Z,#F134
```

```
F13A 3A085C      LD    A,(#5C08)
F13D FE5E        CP    #5E        ;Tecla de BREAK
F13F C8          RET   Z
F140 FE0D        CP    #0D        ;Linea nueva
F142 282B        JR    Z,#F16F
F144 FE0C        CP    #0C        ;Retrocede una posicion
F146 2812        JR    Z,#F15A
F148 FE06        CP    #06        ;Tecla CAPS LOCK
F14A 2819        JR    Z,#F165
F14C D620        SUB   #20        ;Encuentra direccion
F14E 6F          LD    L,A        ; del caracter introducido
F14F 2600        LD    H,00
F151 29          ADD   HL,HL
F152 29          ADD   HL,HL
F153 29          ADD   HL,HL
F154 0100FC      LD    BC,#FC00
F157 09          ADD   HL,BC
F158 0C          INC   C
F159 C9          RET

F15A 3E08        LD    A,08       ;Introduce un retoceso de cursor
F15C D7          RST   #10
F15D 3E20        LD    A,#20      ;Espacio
F15F D7          RST   #10
F160 3E08        LD    A,08       ;Retroceso de cursor
F162 D7          RST   #10
F163 180B        JR    #F170
F165 3A6A5C      LD    A,(#5C6A)
F168 EE08        XOR   08         ;Quita o pone mayusculas
F16A 326A5C      LD    (#5C6A),A
F16D 18C1        JR    #F130
F16F D7          RST   #10
F170 E1          POP   HL         ;Recupera Dir. de retorno
F171 C300F1      JP    #F100
```

Las primeras cuatro instrucciones forman una pequeña rutina que espera hasta que pulses una tecla. Esta es una rutina en código máquina alternativa a la instrucción en BASIC "PAUSE 0", la cual usaremos en próximos programas de demostración. Esta rutina lo primero que hace es poner a cero el bit 5 de la variable del sistema FLAGS, que está en la dirección "5C3Bh" (23611d). Como el registro IY del Spectrum contiene casi siempre la dirección "5C3Ah", todas las variables del sistema pueden ser direccionadas como desviaciones respecto a esta base.

Después de esto, el bit 5 de "FLAGS" será inspeccionado en repetidas ocasiones. Si durante este tiempo no has pulsado ninguna tecla, el bit 5 permanecerá a cero, pero tan pronto como pulses una tecla este bit cambiará a "1" y la instrucción "JRZ" será desechada, por lo que el registro "A" se cargará con el contenido de la variable "LAST_K" (5C08h).

Esto que hemos visto es otra forma que tienes de acceder al teclado.

Inmediatamente después de esto, la rutina inspecciona en qué caso especial de los cuatro posibles nos encontramos, los cuales se indican mediante cuatro códigos, que serán:

— *5E:* Has pulsado la tecla de BREAK, por lo que, como estará a cero el indicador *(flag)* de cero, cuando la subrutina de entrada retorne a la rutina principal, provocará un RETURN de todo el programa.

— *0D:* Es el código de la tecla ENTER (nueva línea o retorno del carro); provoca el paso a una nueva línea y regresa al comienzo de la rutina principal. Esto se lleva a cabo realizando extracciones de direcciones almacenadas en el *stack* (POP) y haciendo saltos incondicionales.

— *0C:* Es el código correspondiente a DELETE. Cuando pulsas la tecla, la rutina saltará a la dirección F15Ah. En esta ocasión la rutina realizará la marcha atrás del cursor en una posición e imprimirá, en la nueva posición, un espacio para borrar lo que hubiera escrito anteriormente en esa posición y, por último, vuelve a hacer retroceder el cursor una posición para restaurar la posición de impresión. Hecho todo esto, la rutina retornará al comienzo realizando las mismas acciones vistas para el caso anterior.

— *06:* Es el código de CAPS LOCK. La rutina saltará para realizar precisamente esto: fijar o quitar la condición de CAPS LOCK por medio de una operación XOR. Después saltará al principio para ver lo que quieres imprimir.

El resto de la subrutina, desde F14Ch, te deberá resultar conocido. Se calcula la dirección del carácter en el nuevo alfabeto y vuelve con dicha información a la rutina principal.

La rutina principal llama (CALL) a la subrutina en dos ocasiones, una por cada uno de los caracteres que forman la composición de "carácter doble", y con frecuencia estos caracteres se colocan juntos en la UDG. El listado de la rutina principal es:

**Código de entrada de caracteres de cuatro bits. Rutina principal**

```
F100                ORG   #F100
F100 1158FF         LD     DE,#FF58 ;UDG "A"
F103 CD30F1         CALL  #F130   ;Llama a subrutina
F106 C8             RET   Z          ;Sale de programa si
                          indicador de cero esta a uno
F107 0608           LD     B,08
F109 7E             LD     A,(HL)
F10A 07             RLCA
F10B 07             RLCA
F10C 07             RLCA
F10D 07             RLCA
F10E 12             LD     (DE),A
F10F 13             INC   DE
F110 23             INC   HL
F111 10F6           DJNZ  #F109
F113 3E90           LD    A,#90
F115 D7             RST   #10        ;Imprime UDG "A"
F116 3E08           LD     A,08
F118 D7             RST   #10        ;Retrocede el cursor
F119 1158FF         LD     DE,#FF58
F11C CD30F1         CALL  #F130   ;Llama a subrutina
F11F C8             RET   Z          ;Sale de rutina si Z=1
F120 0608           LD     B,#08
F122 1A             LD     A,(DE)
F123 B6             OR     (HL)      ;Combina dos caracteres
F124 12             LD     (DE),A
```

```
F125 13          650        INC    DE
F126 23          660        INC    HL
F127 10F9        670        DJNZ   #F122
F129 3E90        680        LD      A,#90
F12B D7          690        RST    #10       ;Imprime UDG "A"
F12C 18D2        700        JR      #F100
```

Las dos mitades de la rutina principal comienzan de la misma forma, tomando la dirección de UDG "A" y almacenándola en DE. Después ambas llaman a la subrutina.

La subrutina está organizada de forma que sólo retorna al programa principal con el bit indicador de cero puesto a uno, si se ha pulsado la tecla de BREAK. En el resto de las situaciones este indicador estará a cero. La instrucción "INC C", en F158h, tiene como propósito el asegurar que el indicador de cero está fijo a cero antes de retornar al programa principal.

Si el indicador de cero está puesto a uno, la instrucción "RET Z" provoca la parada de todo el programa.

Si todo se realiza de una forma normal, la primera parte del programa principal coloca nuestro carácter de cuatro bits en la parte izquierda del espacio de un carácter normal y lo carga UDG "A". No hay ninguna razón especial para haber escogido el espacio de UDG "A"; lo mismo podría haber sido UDG "U" o cualquier otra UDG. El programa imprime entonces el nuevo carácter e, inmediatamente después, realiza un retorno del cursor en una posición para restablecer la posición de impresión sobre esa zona de 8 × 8 aún no completada del todo, ya que sólo estará rellena una mitad.

En la segunda mitad del programa se trabaja con la entrada del próximo carácter; introducimos los ocho bytes de UDG "A" (con el primer carácter de cuatro bits) sucesivamente en el registro A y realizamos con cada uno de ellos una operación "OR", con el contenido de la dirección indicada por el par de registros HL (HL direccionarán cada vez a uno de los ocho bytes que componen el nuevo carácter a introducir), cargándose el resultado en la dirección indicada por el par de registros DE, es decir, otra vez en UDG "A". Realizado esto, se vuelve a realizar la impresión de UDG "A" que, ahora sí, ya ocupará un espacio completo de 8 × 8 bits. Por último, se retorna al principio para volver a comenzar la secuencia.

El efecto que observarás en la pantalla será que se realiza la impresión, cada vez, del carácter que deseas. Las operaciones de borrado y nueva línea trabajan casi como lo hacen normalmente, excepto en el caso del borrado (DELETE), que te borrará dos caracteres cada vez, ya que esta operación trabaja con bloques de 8 × 8 bits, es decir, con posiciones de impresión completas.

## Operaciones lógicas

La operación de suma lógica (OR) es realizada por el microprocesador; constituye una forma limpia y clara de combinar dos caracteres en una misma posición de impresión. Dos letras, cuando ya están preparadas para ser colocadas juntas, es el ejemplo que se ve en la figura 6.10.

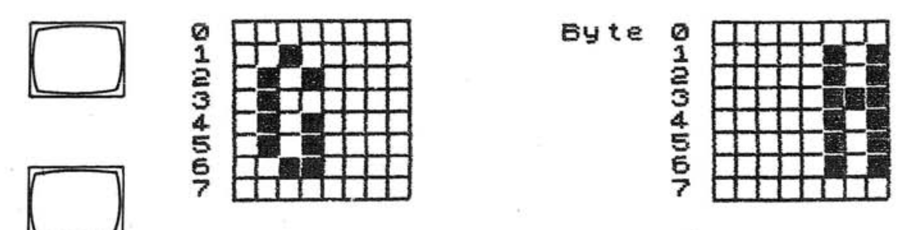



*Fig. 6.10*

La letra "G" ha sido desplazada a la parte izquierda por las operaciones F10Ah a F10Dh. Este será ahora el contenido de la UDG "A", direccionada mediante los registros DE. La letra "H" es el nuevo carácter que has tecleado y es direccionado mediante los registros HL. El registro A contendrá la letra "G" con un byte cada vez, y lo suma, lógicamente, con el byte de la letra "H" direccionado por los registros HL.

Los dos primeros bytes de ambas letras no tienen mucha información al ser todos ceros. Por esto vamos a ver cómo funciona usando el byte 1:

El registro "A" contendrá:
0010 0000

La posición de memoria direccionada por HL contendrá
0000 0101

La operación OR aplicada entre estos dos bytes dará como resultado, en el registro A: 0010 0101.

Esto será cargado en la dirección que indiquen los registros DE, para así reemplazar el carácter existente. Como apunte de interés vamos a analizar qué habría pasado si hubiésemos hecho la operación "AND" en vez de la "OR", entre el contenido del registro A y la posición que indica HL. En este caso el resultado habría sido "0000 0000".

Cuando todos los bytes de ambos caracteres han sido relacionados mediante operaciones "OR", el resultado será el de la figura 6.11.

Como estamos hablando de operaciones lógicas, estudiaremos ahora la operación XOR ("O exclusiva"), la cuál he utilizado anteriormente para fijar a uno o cero la función CAPS LOCK.




*Fig. 6.11*

La operación de la que te hablo fue:

**Set/Reset CAPS LOCK**

```
F165            10      ORG    #F165
F165 3A6A5C     20      LD      A,(#5C6A)
F168 EE08       30      XOR    #08
F16A 326A5C     40      LD      (#5C6A),A
```

En la variable del sistema FLAGS 2, en la dirección 5C6Ah (23658d), hay una agrupación de indicadores *(flags)*. El bit 3 de este byte controla el estado de CAPS LOCK, estando a uno, si la función está activa, y a cero si no lo está. Suponemos que hasta ahora la función no está activa, por lo que el byte que contiene el indicador de CAPS LOCK tendrá una configuración como la que sigue:

xxxx 0xxx

Las "x" indican que este bit puede estar en este momento a cero o a uno, ya que en este momento no nos importa cómo estén.

Ahora vamos a realizar la función "O exclusiva" (XOR) de esta variable, con el valor 08h:

0000 1000

La función "O exclusiva", debido a su forma de trabajar, coloca un bit a uno cuando se realiza la operación entre bits que no son iguales (es decir, no son los dos cero o uno) y lo coloca a cero cuando los dos bits son iguales. Aplicando esto, el resultado será:

xxxx 1xxx

Cargaremos este resultado en la variable del sistema FLAGS 2 y estaremos trabajando con mayúsculas (CAPS LOCK a uno).

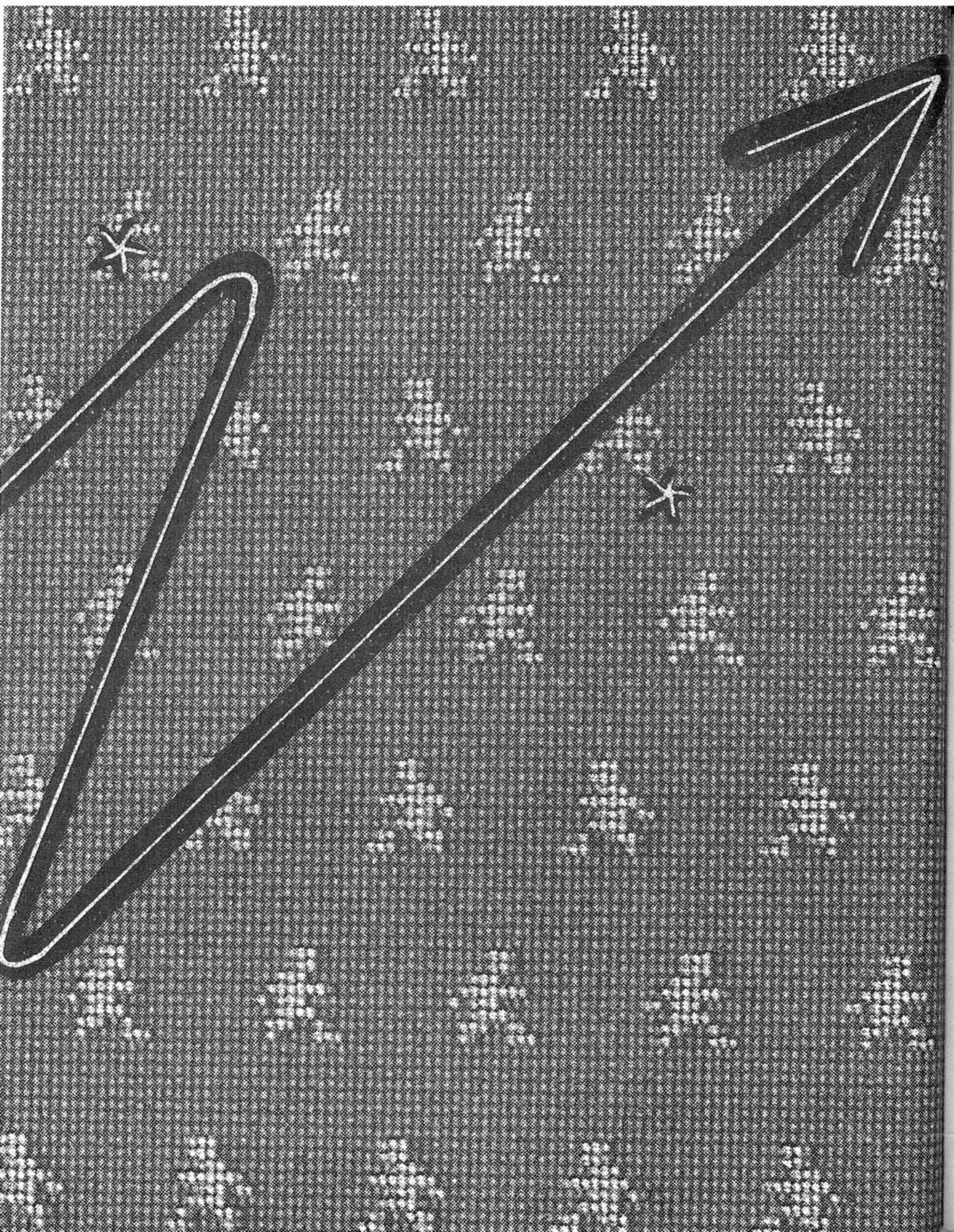
Vamos a ver qué ocurrirá si efectuamos la operación anterior por segunda vez. Analizando el estudio que hicimos de la operación "O exclusiva" te será fácil deducir que ahora el bit 3, al realizar la operación con 08h, quedará puesto a cero. Es decir, el resultado final será:

xxxx 0xxx

con lo que volveremos a trabajar con minúsculas.

Los bits "x" permanecerán siempre inalterados durante estas operaciones, ya que al aplicarse la operación "O exclusiva" estos bits la realizan con los ceros del byte 08h, por lo que, según se explicó, la función "O exclusiva" no les afectará.

Como puedes deducir, la operación "O exclusiva" es una operación muy útil para modificar el valor de un conmutador de funciones.



# 7
# Generación de caracteres de seis bits

La realización de textos con letras de seis bits, para poder imprimir 40 caracteres por línea, es bastante más complicada que la analizada en el capítulo anterior. Estos caracteres de seis bits ocupan tres cuartas partes del espacio reservado para un carácter normal de 8 × 8 bits. En la figura 7.1 puedes ver de una forma gráfica lo que te estoy explicando.

La figura 7.1 muestra las letras tal y como están almacenadas en el nuevo alfabeto de seis bits.

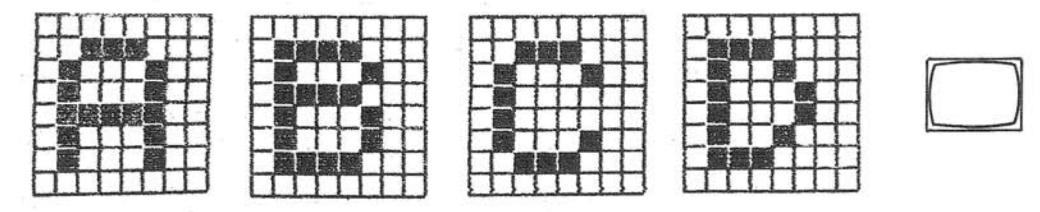


*Fig. 7.1*

Tenemos que reordenar las letras formando bloques con cuatro letras cada uno, tal y como se muestra en la figura 7.2.

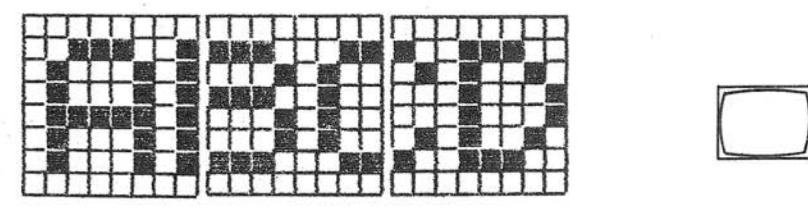


*Fig. 7.2*

En primer lugar, debes observar que no hay una forma clara en el Spectrum de que puedas imprimir, de una forma directa, un carácter desplazado a la derecha o a la izquierda dentro del espacio de su posición de impresión. Las posiciones de impresión son colocadas en la pantalla de una forma rápida y con un formato fijo.

Para imprimir un carácter desplazado lateralmente, dentro de su espacio de impresión, has de colocar el carácter repartido entre dos posiciones de impresión contiguas, con lo que has de generar dos nuevos caracteres en la memoria, uno que contenga la parte izquierda del carácter original y otro que contenga la parte derecha del carácter. Hecho esto, si realizas la impresión sucesiva de ambos caracteres, te aparecerá en la pantalla un carácter como el original, pero desplazado lateralmente.

En el caso de la figura, puedes ver que la letra "B" ha sido desplazada a la izquierda en dos bits, la letra "C" está desplazada cuatro bits y la letra "D" en seis bits.

Hay, por tanto, tres caracteres compuestos (es decir, tres posiciones de impresión) que contienen los cuatro caracteres originales.

Vamos a analizar el proceso de desplazamiento necesario para mover los caracteres; pero antes ya habrás observado que es necesario que nos planteemos el diseño de algún tipo de contador que controle el número de bits a desplazar (dos, cuatro o seis).

## Diseño de un contador

Vamos a plantearnos unas consideraciones iniciales sobre el contador. Debido a que la rutina va a ser llamada cada vez que se quiera realizar una letra, el contador tendrá que estar situado en una posición de la memoria en la que siempre podamos conocer su contenido, pero, al mismo tiempo, donde no lo perdamos cada vez que retornemos al programa en BASIC.

Una dirección que responde a estos requisitos es la de la variable del sistema situada en la dirección 5C81h (23681d), la cual no es utilizada por el Sinclair. Podemos inicializar el contador antes de que comencemos a utilizarlo en la rutina de generación del carácter y organizarlo para que se realicen dos adiciones en él cada vez que se utilice en la rutina. La rutina también debe chequear en qué momento el valor del contador es mayor de seis para, en este caso, poner de nuevo a cero dicho byte.

Una pequeña parte de la rutina que hace esto, podría ser la siguiente:

```
892C 21815C         LD     HL,#5C81
892F 34             INC   (HL)
8930 34             INC   (HL)
8931 7E             LD     A,(HL)
8932 CB5F           BIT   3,A
8934 CA00F0         JP    Z,#F000 ;Salta a proxima operacion
8937 CB9E           RES    3,(HL)
```

Observa cómo se controla el momento en que el contador ha pasado de seis, para, en ese momento, ponerlo a cero de nuevo. Cuando la rutina añade dos a un seis en el byte direccionado por HL, éste pasará a valer ocho, lo que quiere decir que se pone a uno el bit 3 de dicho byte por primera vez, poniendo, al mismo tiempo, los bits más bajos a cero (0000 1000). Cargando el registro A con dicho byte, direccionado por HL, y

comprobando el estado del bit 3, podemos controlar cuándo dicho byte pasa a valer ocho. En el momento en que se haya llegado a ocho, bastará con poner a cero el bit 3 para poner a cero el contador. Puedes comprobar el estado del bit 3 directamente sin tener que pasar el byte por el registro A, pero es necesario utilizar el registro A para el contador, por lo que he preferido aprovechar esto y comprobar el bit 3 cuando está en el registro A.

## Desplazamiento de las letras

El trabajo que supone desplazar las letras a su nueva posición de impresión resulta muy sencillo pero laborioso a su vez, ya que hay que trabajar con bastantes instrucciones en código máquina.

En la realización de este trabajo necesitarás utilizar tres caracteres del UDG para operar con ellos (UDG 1, 2 y 3, que se corresponden con "A", "B" y "C"). Para empezar, resulta evidente que la rutina trabajará con un byte cada vez, pero el principio de su funcionamiento resulta más fácil de comprender si se analiza el mismo con todos los bytes que forman un carácter.

Si la letra "A" es la primera del bloque de cuatro y la letra "B" es la segunda, en primer lugar has de colocar la letra "A" en el UDG "1" y la "B" en el UDG "2". Una vez hecho esto, has de comenzar a desplazar la letra "B" hacia la izquierda realizando un desplazamiento aritmético del UDG "2", el cual coloca un cero en el bit 0 e introduce el bit 7 en el bit de acarreo. Después de esto, extraes el bit de acarreo por medio de una operación de desplazamiento hacia la izquierda y lo pasas hacia los bytes en blanco de UDG "3". Después de hacer esto dos veces (para todos los bytes del carácter), los dos nuevos caracteres tendrán el aspecto que se muestra en la figura 7.3.

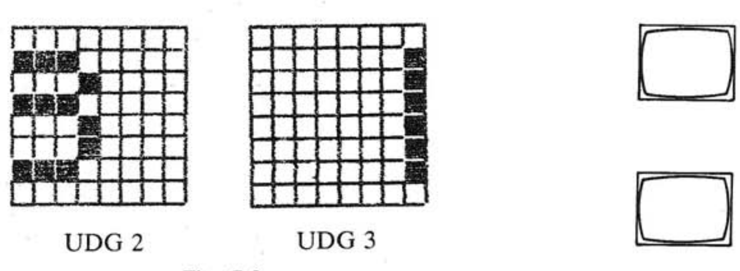



*Fig. 7.3*

Ahora puedes realizar la operación OR entre UDG "3" y UDG "1", byte a byte, y habrás completado la formación de tu primer carácter compuesto (situado en UDG "1"); también tendrás la letra "B" colocada en la posición final que ocupará en UDG "2". El resultado gráfico será el que aparece en la figura 7.4.

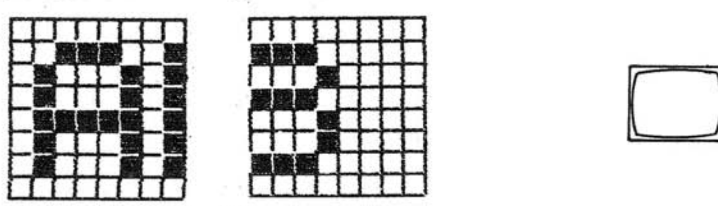


*Fig. 7.4*

Faltan aún dos cosas más por hacer antes de que puedas mover la siguiente letra al segundo carácter, situado en UDG "2". En primer lugar, has de hacer retroceder el cursor de impresión de forma que quede apuntando a la posición de impresión de la UDG "2". La impresión se realizará siempre en bloques de dos caracteres, pero cada vez que imprimas un bloque, tendrás que retroceder el cursor una posición.

En segundo lugar, has de colocar el contenido actual de la UDG "2" en la UDG "1" (liberada ya del anterior carácter compuesto), dejando la UDG "2" libre para la próxima letra.

Al comienzo de la siguiente pasada de la rutina de impresión, ésta se encontrará con las dos UDG's con un aspecto como el que se muestra en la figura 7.5.

Ahora ya puedes desplazar la letra "C", utilizando los mismos métodos que antes, pero esta vez el desplazamiento ha de ser de cuatro bits en vez de dos como anteriormente.

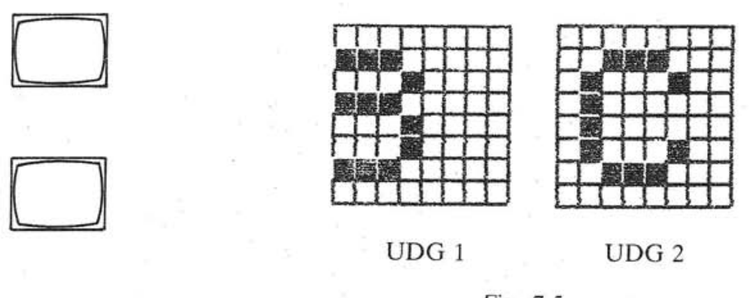



*Fig. 7.5*

Una vez hecho esto, puedes utilizar el mismo sistema para colocar la letra "D", haciendo esta vez desplazamientos de seis bits; después de realizar todo lo anterior, el bloque de impresión está terminado y podrás comenzar a ejecutar de nuevo la rutina con otro bloque.

El siguiente listado es el de la rutina anteriormente analizada.

## Generación de letras de seis bits

```
FBA0                ORG   #FBA0
FBA0 3A085C         LD     A,(#5C08) ;Calcula la Dir. de
FBA3 D620           SUB   #20       ; ultimo caracter en
                           nuevo alfabeto
FBA5 6F             LD     L,A
FBA6 2600           LD     H,00
FBA8 29             ADD   HL,HL
FBA9 29             ADD   HL,HL
FBAA 29             ADD   HL,HL
FBAB 0100FC         LD     BC,#FC00
FBAE 09             ADD   HL,BC

FBAF 1160FF         LD     DE,#FF60 ;Carga nuevo caracter
FBB2 010800         LD     BC,#0008 ; en UDG 2
FBB5 EDB0           LDIR
```

```
FBB7 0608        160       LD     B,#08    ;Limpia espacio de UDG3
FBB9 AF          170       XOR    A
FBBA 12          180       LD     (DE),A
FBBB 13          190       INC    DE
FBBC 10FC        200       DJNZ   #FBBA
                 210
FBBE 21815C      220       LD     HL,#5C81
FBC1 34          230       INC    (HL)     ;Incrementa el contador
FBC2 34          240       INC    (HL)
FBC3 7E          250       LD     A,(HL)
FBC4 CB5F        260       BIT    3,A
FBC6 2802        270       JR     Z,#FBCA
FBC8 CB9E        280       RES    3,(HL)
FBCA 2160FF      290       LD     HL,#FF60
FBCD 2022        300       JR     NZ,#FBF1
                 305
FBCF 0608        310       LD     B,08     ;Desplaza laposicion
FBD1 1E68        320       LD     E,#68    ; del caracter entre
FBD3 F5          330       PUSH   AF       ; UDG2 y UDG3
FBD4 CB26        340       SLA    (HL)
FBD6 EB          350       EX     DE,HL
FBD7 CB16        360       RL     (HL)
FBD9 EB          370       EX     DE,HL
FBDA 3D          380       DEC    A
FBDB 20F7        390       JR     NZ,#FBD4
FBDD F1          400       POP    AF
FBDE 23          410       INC    HL
FBDF 13          420       INC    DE
FBE0 10F1        430       DJNZ   #FBD3
                 440
FBE2 EB          450       EX     DE,HL    ;Realiza la operacion
FBE3 2E58        460       LD     L,#58    ; OR entre los caracteres
FBE5 0608        470       LD     B,08     ; UDG1 y UDG2
FBE7 1A          480       LD     A,(DE)
FBE8 B6          490       OR     (HL)
FBE9 77          500       LD     (HL),A
FBEA 23          510       INC    HL
FBEB 13          520       INC    DE
FBEC 10F9        530       DJNZ   #FBE7
                 540
FBEE 3E90        550       LD     A,#90    ;Imprime UDG1
FBF0 D7          560       RST    #10
FBF1 3E91        570       LD     A,#91    ;Imprime UDG2
FBF3 D7          580       RST    #10
FBF4 3E08        590       LD     A,08     ;Retorna el cursor una posicion
FBF6 D7          600       RST    #10
                 610
FBF7 1E58        620       LD     E,#58    ;Pasa UDG2 a UDG1
FBF9 0E08        630       LD     C,08
FBFB EDB0        640       LDIR
FBFD C9          650       RET
```

La secuencia de operaciones en la rutina es la desarrollada en los párrafos anteriores, pero hay ciertas suposiciones de trabajo, en los direccionamientos, sobre los que tal vez sea necesaria alguna aclaración.

## Almacenamiento, en memoria, de los bytes

Lo primero a señalar acerca de las direcciones es que la mayoría de ellas se refieren a tres de los caracteres redefinibles por el usuario (UDG "1", "2" y "3"). Las direcciones

de estos caracteres comienzan con "FF..". Estas direcciones están contenidas en los registros HL y DE, utilizándolas para las transferencias de los caracteres en sus distintos estados de formación. Esto quiere decir que, una vez que estos registros han sido inicializados, el primer byte de la dirección (el byte en H y D) permanece siempre inalterado, por lo que, en vez de utilizar las instrucciones de tres bytes "LD HL,xxxx" y "LD DE,xxxx", podemos utilizar las instrucciones de dos bytes "LD L,xx" y "LD E,xx".

En la posición FBB7h se encuentra una operación de almacenamiento de un byte en la cual se limpia el contenido del UDG 3, para dejarlo listo para recibir nuestro nuevo carácter. El UDG 3 comienza en la dirección FF68h, y como el registro DE contiene esta dirección al acabar la operación LDIR de la sección anterior, no es necesario que volvamos a direccionar un registro para realizar la operación de limpieza.

Al acabar la dirección de desplazamiento, el registro DE también apuntará al UDG 3. En ese momento, esta dirección pasa a estar contenida en HL para trabajar en la siguiente sección de la rutina, para lo que utilizamos la operación "EX DE,HL", después de lo cual podemos volver a cargar el registro DE con una nueva dirección.

En la posición FBF7h los dos registros HL y DE permanecen inalterados por las operaciones "RST 10". El registro HL apuntará ahora al UDG 2 (después de acabarse la sección anterior), por lo que sólo tendrás que cambiar el registro E para obtener las dos direcciones que se necesitan para la transferencia final.

La forma en que se realiza el control y la puesta a cero del contador es fácil de comprender, ya que sólo hay ligeras variaciones con respecto al método básico ya explicado. El contador está situado en la posición 5C81h. En primer lugar, se aumenta su valor en dos unidades y se pasa su contenido al registro A. Ya en el registro A, se comprueba el estado de su bit 3 para saber si el contador es mayor o igual a ocho. Si el bit 3 está a uno, el número del contador es ocho y el indicador de cero se pondrá a cero al ejecutar la instrucción "BIT 3,A", en esta situación; por el contrario, si el bit 3 del contador está a cero, la operación "BIT 3,A" pondrá a uno el indicador de cero.

Este indicador de cero se utiliza en la rutina en dos saltos condicionales, los cuales utilizan uno u otro camino a través de la rutina, según se necesite en cada caso.

En el primer caso, si el indicador de cero queda a cero, el primer salto condicional situado en la posición FBC6h queda omitido. La rutina pasa a ejecutar las instrucciones "RES 3,(HL)" y "LD HL,FF60". Ninguna de estas dos operaciones nos influye en los indicadores del estado de la máquina, aunque en un primer momento podrías haber pensado que la operación de RESET podría hacerlo; por tanto, el indicador de cero estará aún fijo en el estado en que estaba antes de ejecutar estas instrucciones y provocará, por tanto, la ejecución del salto condicional a "PRINT UDG 2", situada en la posición "FBF1h". Cuando la rutina vaya a realizar la última transferencia, el registro HL ya contendrá la dirección adecuada.

En el otro caso, cuando el indicador de cero es puesto a uno (es decir, que el contenido del contador es menor que ocho), los saltos condicionales se ejecutarán al contrario que en el caso anterior. El salto "JR Z", de la posición "FBC6h", provocará un salto a la instrucción "LD HL,FF60", situada en la posición "FBCAh", pasando por alto la operación de RESET. En esta situación el salto "JR NZ", de la posición FBCDh, será ignorado por la rutina, la cual continuará con la operación de desplazamiento del carácter.

Esta doble combinación de saltos es necesaria, ya que ambas secciones de la rutina ne-

cesitan que haya la misma dirección en el registro HL, pero no podemos realizar la carga de dicho registro antes de comprobar el estado del bit 3 del contador, ya que este registro contendrá la dirección del byte al cual pertenece el bit que estamos comprobando.

Con este método no es necesario añadir ninguna instrucción al programa, el cual nos almacenará tres bytes por cada instrucción "LD HL,xxxx".

Todos estos controles de los contenidos de los registros, en cada momento, no son absolutamente necesarios, pero esto nos permite ahorrar muchos bytes de memoria. En el presente caso, este método nos permite ahorrar alrededor de 16 bytes sobre el espacio que ocuparía una rutina que utilizara los direccionamientos totales; esto es equivalente a un ahorro de aproximadamente el 15 por 100. Esto se realiza ejecutando el programa en su versión inicial, más extensa, y observando dónde merece la pena realizar una compresión. Esto resulta bastante complicado y cansado.

## Utilización de los caracteres de seis bits

Aunque ya hayas conseguido, al haber llegado aquí, la realización de una línea con 40 caracteres, existen ciertas restricciones en la utilización de este método de impresión. Aunque es posible incluir la posibilidad de que puedas meter comandos de «nueva línea», tal y como lo hiciste para el caso de la rutina para la generación de caracteres de cuatro bits, no hay un método satisfactorio para realizar el desplazamiento del cursor hacia atrás ni para corregir un error de edición, debido a que las letras se reparten entre posiciones distintas de impresión. En realidad, yo no veo una gran utilidad en el uso de esta técnica para visualizaciones en la pantalla, cuando estamos trabajando como si fuera una máquina de escribir.

La aplicación práctica de los caracteres de seis bits creo que está ligada a la impresión de etiquetas o de textos. Por ejemplo, si tienes un programa para crear un listín de teléfonos o de direcciones, las nuevas indicaciones pueden ser introducidas de una forma más clara y económica usando el nuevo alfabeto de caracteres de seis bits. Esto quiere decir que podrías recuperar e imprimir cadenas de caracteres que tuvieras almacenadas y para las que no surgen las necesidades de corrección. La figura 7.6 da una idea de lo que gana el texto en aspecto compacto.

```
ES UNA LINEA DE 32 CARACTERES DE
 8-BITS**
ES UNA LINEA DE 40 CARACTERES DE 6-BITS**
```

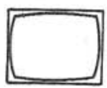


*Fig. 7.6*

Para acceder a la rutina desde una cadena de caracteres ya existente, te resultará muy útil un programa en BASIC como el que te doy a continuación.

**Impresión en seis bits (demostración)**

```
  10 LET w$="Es una linea de 40
caracteres de 6-BITS*"
  20 PRINT AT 9,0;w$
```

```
   30 POKE 23681,6
   40 FOR j=1 TO LEN w$: RANDOMIZ
E USR 64416
   60 NEXT j
```

La instrucción POKE de la línea 30 inicializa el contador a un valor de seis. Esto significa que la rutina imprimirá la primera letra como el comienzo de un bloque de cuatro en la posición de la letra "A" en la figura 7.2.

Esta nueva sugerencia para realizar la impresión no tendría una utilización muy práctica si no pudieras utilizar la impresora. Una vez más, esto se puede realizar, pero has de tener un gran cuidado al manejarlo.

El problema es que el comando LPRINT, el cual vuelca el contenido del *buffer* de la impresora, parece incapaz de realizar impresiones en las que se utiliza la reposición del cursor en una posición anterior, así como otra serie de particularidades necesarias para la impresión de pares de UDG's. La solución consiste en preparar antes una línea completa en el fichero de imagen *(display file)* y después volcarlo a la impresora, utilizando para ello una modificación del comando COPY.

Tal y como está en la ROM del Sinclair, la rutina COPY vuelca toda la pantalla a la impresora, de línea de imagen en línea de imagen, para así conseguir líneas consecutivas, en vez de utilizar la difícil organización interna del fichero de imagen (véase capítulo 9).

En nuestro caso sólo queremos volcar a la impresora ocho líneas de imagen (barridas del haz de la pantalla), por lo que el redireccionamiento necesario es mínimo. Podemos dividir la rutina de COPY residente en la ROM (la cual comienza en la dirección "0EACh"), para así realizar una versión sencilla que extraiga una sola línea de imagen de la pantalla y la vuelque a la impresora.

La línea de impresión de caracteres de seis bits ha de ser colocada en la pantalla antes de hacer el COPY a la impresora; y tal vez el mejor lugar de la pantalla para colocarla sea la parte baja de ésta, al no molestar a la zona de trabajo de la pantalla. Esto se consigue incluyendo "PRINT # 1;" en el programa en BASIC anterior.

La rutina de COPY resultante es:

**Impresión en la línea 1 de la parte baja de la pantalla**

```
FF00              10       ORG    #FF00
FF00 F3           20       DI
FF01 0608         30       LD      B,08
FF03 21E050       40       LD      HL,#50E0 ;Dir. linea 1 parte inferior
                  50 ;                de la pantalla
FF06 E5           60       PUSH   HL
FF07 C5           70       PUSH   BC
FF08 CDF40E       80       CALL   #0EF4 ;Llama subrutina de impresion
FF0B C1           90       POP    BC
FF0C E1          100       POP    HL
FF0D 24          110       INC    H
FF0E 10F6        120       DJNZ   #FF06
FF10 CDDA0E      130       CALL   #0EDA ;Llama ultimaparte de
                 140 ;               rutina para copiar
```

```
FF13 0602          LD    B,02
FF15 CD440E        CALL  #0E44 ;Llama a rutina para
                   ;            limpiar linea
FF18 C9            RET
```

La rutina comienza con una instrucción que anula la posibilidad de interrumpirla (DI), tal y como se realiza en la rutina COPY original. Se crea un contador de ocho dígitos en el registro B y carga el registro HL con la dirección de comienzo de la primera línea de la parte baja de la pantalla, la cual es 50E0h. La rutina actual de impresión, que vuelca solamente una línea desde la pantalla a la impresora, se encuentra en la dirección 0EF4h. La instrucción "INC H" actualiza la dirección contenida en el registro HL para que apunte a la próxima línea de pantalla. La rutina vuelve a la posición "0EDAh" de la ROM, para realizar la parte final de la rutina COPY residente en la ROM. Esta última parte realiza la representación en la impresora y finaliza la rutina. Las dos últimas instrucciones antes de RET se realizan para dejar en el estado original las variables usadas. Estas dos instrucciones limpian las dos últimas líneas de la pantalla, dejándolas listas para otra impresión.

La rutina que se utiliza para realizar esto es "CL_LINE", residente en la posición "0E44h" de la ROM. Realiza la limpieza de los números de línea especificados en el registro B, de 1 a 24, escritos en la parte baja de la pantalla. Es una rutina muy útil como ayuda a la rutina más completa, CLS, que se encuentra en la posición "0D6Bh".

Suponiendo que tienes el código agrupado tal y como se muestra a continuación, entonces está ya preparado para la parte final del programa en BASIC.

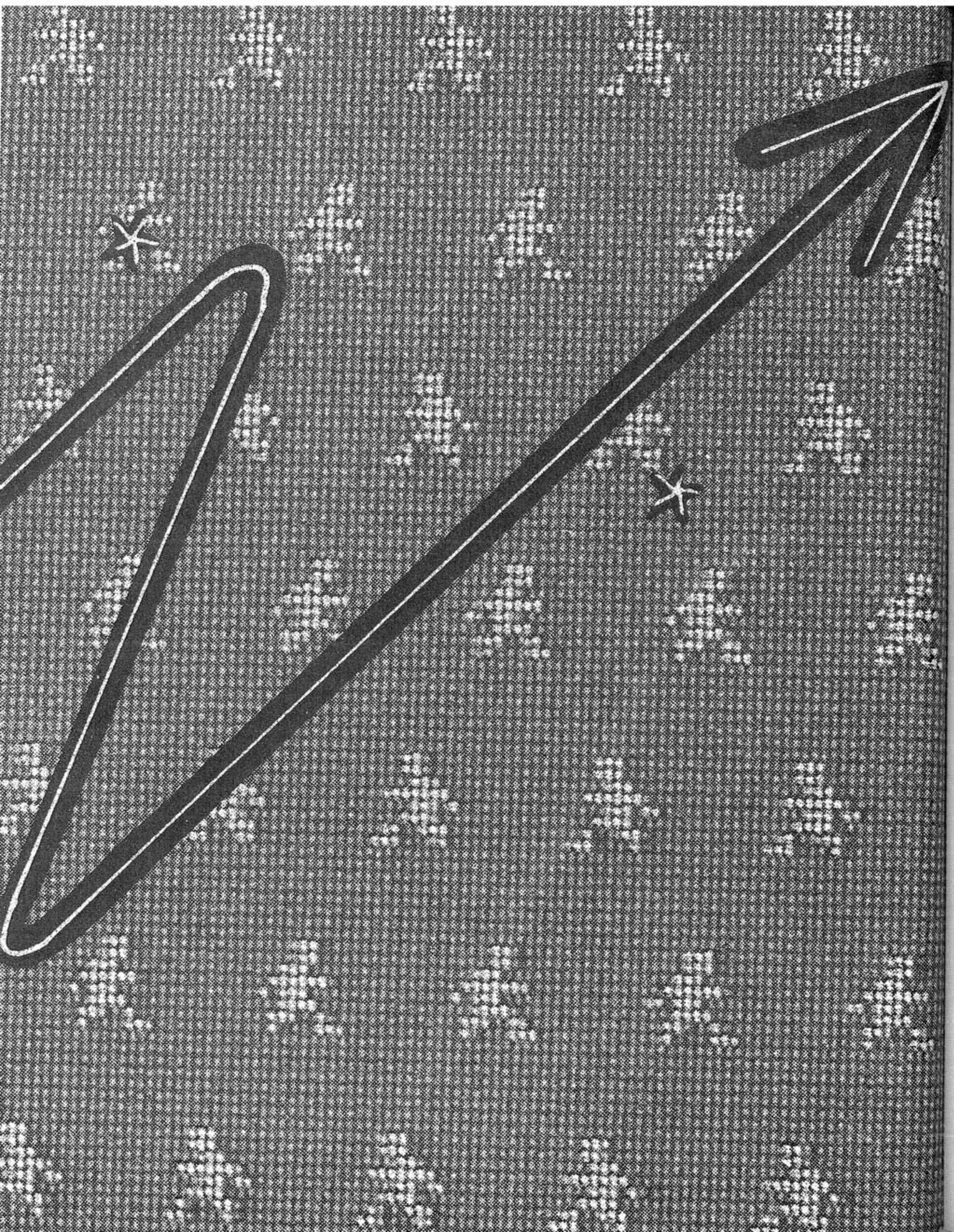
| | | |
|---|---|---|
| FBA0-FBFD | impresión en seis bits | comienzo en "64416d" |
| FC00-FEFF | alfabeto | |
| FF00-FF18 | COPY | comienzo en "65280d" |

El programa en BASIC de demostración será:

**Impresión de caracteres de seis bits**

```
  10 LET w$="Es una linea de 40
caracteres de 6-BITS*"
  20 PRINT AT 9,0;w$
  30 POKE 23681,6
  40 FOR j=1 TO LEN w$(j): RANDO
MIZE USR 64416
  60 NEXT j
  70 RANDOMIZE USR 65280
```

Si te molesta el que las líneas de texto de la parte baja de la imagen aparezcan con la opción FLASH, puedes añadir "INK 7;" al final de la línea 30.



# 8
# Animación. Figuras móviles

Todo el mundo ha de conocer lo que son los *sprites:* un grupo de bytes que normalmente forman una imagen en la pantalla y pueden moverse por toda ella, independientemente de cualquier otra cosa que pueda verse en la pantalla.

Los gráficos móviles más refinados están controlados por un circuito especial, el cual trata a todos los bytes que forman la imagen móvil *(sprite)* como a un bloque; esto quiere decir que el programador sólo debe indicar lo que ha de moverse y en qué dirección lo hará dentro de la pantalla. El Spectrum no posee tal circuito, por lo que los movimientos de los gráficos han de realizarse bajo el control de programas.

Hay tres tipos de movimiento que hemos de tener en cuenta: movimiento en el interior del gráfico movil (animación); movimiento del gráfico móvil por la pantalla, y movimiento por delante o detrás del gráfico móvil considerado, de otros gráficos móviles o del fondo.

En este capítulo trabajaremos con el primer tipo de movimiento (movimiento en el interior del gráfico móvil).

## Realización de gráficos móviles

Para cualquier tipo de animación necesitas preparar una serie de figuras en la pantalla, cada una ligeramente diferente de la anterior, por lo que, cuando un ciclo de representación está en marcha de forma continuada, te dará la impresión de que el gráfico está animado, se mueve.

Es posible realizar la animación en el interior del espacio de un carácter; los caracteres del UDG son muy manejables para este tipo de trabajos, así como para otros traba-

jos, como ya hemos podido comprobar. A continuación puedes ver un ejemplo en el que se trabaja con un gráfico que representa una cara pequeña y fea llamada "Snapper", que he utilizado para realizar algunos juegos.

**Valores para la realización de los gráficos del "Snapper"**

| *Cara 1* | *Cara 2* | *Cara 3* | *Cara 4* |
|---|---|---|---|
| 126 | 126 | 126 | 126 |
| 255 | 219 | 255 | 255 |
| 165 | 255 | 153 | 153 |
| 129 | 165 | 255 | 153 |
| 129 | 129 | 165 | 255 |
| 165 | 165 | 165 | 102 |
| 102 | 102 | 102 | 60 |
| 60 | 60 | 60 | 0 |

**Almacenamiento de los datos necesarios para formar el «Snapper»**

```
   5 DATA 126, 255, 165, 129, 12
9, 165, 102, 60, 126, 219, 255,
165, 129, 165, 102, 60, 126, 255
, 153, 255, 165, 165, 102, 60, 1
26, 255, 153,153, 255, 102, 60,
0
  10 FOR j=0 TO 31
  20 READ n: POKE USR "  "+j,n
  30 NEXT j
```

Para ver la apariencia del "Snapper" puedes usar el siguiente programa.

**"Snapper"**

```
100  FOR j=0 TO 3
110 PRINT AT 10,10:CHR$ (144+j)
120 PAUSE 5: NEXT j
130 PAUSE 10: GO TO 100
```

Pero, para poder realizar todos los gráficos que se te puedan ocurrir, normalmente necesitarás más espacio que el que te proporciona un carácter, es decir, necesitas crearte un gráfico móvil *(sprite)*. Para empezar, podemos definir un bloque con un espacio equivalente a nueve caracteres, formando un cuadrado de tamaño 3 × 3 bytes.

Dibujar en el interior de dicho cuadrado es un poco más complicado que hacerlo en el interior del espacio de un carácter. Hay una gran cantidad de programas para dibujar que puedes usar, pero a continuación encontrarás uno sencillo que yo he creado para producir el tipo de cosas que queremos.

**Creación de un "sprite"**

```
  5 LET n=1: LET a=0
 10 LET x=12: LET y=163
 15 PRINT AT 0,(x-12)/8; PAPER
5;" "; PAPER 6;" "; PAPER 5;" "
 16 PRINT AT 1,(x-12)/8; PAPER
6;" "; PAPER 5;" "; PAPER 6;" "
 17 PRINT AT 2,(x-12)/8; PAPER
5;" "; PAPER 6;" "; PAPER 5;" "
 20 PLOT  INVERSE a;x,y
 25 PRINT AT 18,0;"Dibuja trama
 numero ";n'"3=</4=<\9=>/0=>\"
 30 PAUSE 0
 40 LET y$=INKEY$
 45 IF y$="z" THEN  LET a=NOT a
 50 IF y$="5" THEN  LET x=x-(x>
0)
 51 IF y$="8" THEN  LET x=x+(x<
23)
 52 IF y$="6" THEN  LET y=y-(y>
152)
 53 IF y$="7" THEN  LET y=y+(y<
175)
 54 IF y$="9" THEN  LET x=x+(x<
23): LET y=y+(y<175)
 55 IF y$="0" THEN  LET x=x+(x<
23): LET y=y-(y>152)
 56 IF y$="4" THEN  LET x=x-(x>
0): LET y=y+(y<175)
 57 IF y$="3" THEN  LET x=x-(x>
0): LET y=y-(y>152)
 58 IF y$=CHR$ 13 THEN  GO TO 1
00
 60 GO TO 20
```

Cuando el programa se está ejecutando, el cursor imprimirá una línea con un solo punto de imagen cada vez *(pixel):* a la derecha, izquierda, arriba y abajo. Las teclas "3", "4", "9" y "0" realizan las líneas en diagonal, tal y como se puede comprobar.

Las líneas 15, 16 y 17 imprimen unas barras de control en azul y amarillo que te ayudan a encontrar dónde te encuentras cuando estás dibujando. Las comparaciones (X>0), (Y<175), etc., son para evitar que los puntos de imagen a dibujar se salgan del cuadrado de la pantalla. Estas expresiones tomarán valor "1" si son ciertas y "0" si son falsas, por lo que sólo se cambiará el valor de las variables X e Y si ambas están dentro de los límites especificados. En la línea 58, "13" es el código de ENTER, el cual finaliza esta parte del programa.

Tecleando "z" se realiza la inversión del comando PLOT (línea 45). Esto quiere decir que con el cursor se pueden borrar los puntos de imagen que ya hayas dibujado anteriormente. Esta inversión se mantendrá hasta que vuelvas a teclear "z" otra vez.

Ahora ya puedes comenzar a dibujar tu propio *sprite.*

Algunas veces supone una ayuda realizar un boceto en papel como avance de lo que será tu dibujo. Cuando hayas terminado tu *sprite,* lo próximo que debes hacer es prepararte para almacenarlo en alguna zona de memoria. Hasta este momento el *sprite* carecerá de movimiento en la pantalla.

A continuación encontrarás un programa que te permitirá almacenar tu *sprite,* byte a byte, en la parte alta de la RAM, comenzando en la dirección "61440d". No ocurre nada especial en esta dirección, únicamente presenta la ventaja de que su valor en hexadecimal es "F000h", lo que significa que el byte menos significativo (LSB) es cero. Como tendrás que sumar direcciones relativas a este número, este LSB a cero te ayudará a comenzar el almacenamiento.

**Almacenamiento del "sprite" en la zona alta de la memoria**

```
   5 LET n=1: LET a=0: LET q=614
40
 100 FOR k=0 TO 2
 110 FOR i=0 TO 2
 120 FOR j=0 TO 7
 130 POKE q+j+8*(i+3*k),PEEK (16
384+32*k+256*j+i)
 140 NEXT j: NEXT i: NEXT k
 150 LET q=q+72
 160 LET n=n+1: IF n<5 THEN  GO
TO 20
```

El programa es muy corto, pero utiliza cuatro variables nuevas que trabajan como veremos a continuación. La dirección de comienzo de cada grupo de *sprites* está almacenada en la variable "q", la cual se aumenta en un valor de "72" en cada pasada del programa (línea 150). Los bucles controlados por las variables "k", "i" y "j" seleccio-

nan cada byte del grupo de una forma ordenada: la variable "j" controla los ocho bytes que forman el espacio de un carácter, la variable "i" selecciona la posición en columnas y la variable "k" posiciona la línea.

El valor "256", que se opera con la variable "j" en la línea 130, se debe a la forma en la cual está organizado el fichero de imagen en el Spectrum (véase el capítulo siguiente). Todos los primeros bytes de las ocho líneas más altas de la pantalla son inspeccionados en primer lugar, después los segundos bytes, y así sucesivamente (véase página 120 del *Manual del Spectrum).* Puede que esta organización se deba a una necesidad creada por la ROM del Spectrum (véase próximo capítulo), pero puede ser un gran problema para los programadores, especialmente cuando trabajan en código máquina. Debes añadir siempre "256" y recordar si te encuentras en la línea de pantalla ocho o en la nueve.

La variable "n" controla el número de *sprites* que estamos generando. Yo he colocado cuatro. El valor mínimo para obtener un efecto razonable de animación es de tres, pero con cuatro se consigue una mayor suavidad entre los cambios en la imagen.

Introduce este programa y colócalo en memoria con el anterior (con la instrucción MERGE, para que este último no borre al anterior). Cuando comiences a ejecutarlo, tendrás la oportunidad de diseñar y almacenar cuatro *sprites.* La segunda parte del programa, la cual almacena los *sprites,* tardará un poco de tiempo en ejecutarse (espera la aparición del próximo número antes de volver a utilizar el cursor).

En este programa, la posición actual del cursor y el anterior modelo creado permanecen almacenados. Muchos de los efectos de animación consisten en la modificación de parte de un diseño, permaneciendo el resto del diseño inalterado; puedes hacer esto usando la opción de borrado.

Si deseas tener una pantalla en blanco cada vez que lo ejecutas, debes cambiar la línea 165 por "GO TO 10".

Diseñar una rutina que te permita observar los gráficos una vez finalizados, es fácil de hacer y fácil de utilizar en conjunción con otros programas. La solución consiste en cambiar la variable del sistema UDG situada en la dirección 5C7Bh (23675d), de forma que apunte al comienzo de los *sprites* cada vez que se quiera visualizar uno de ellos. Los respectivos caracteres gráficos aparecerán almacenados en las posiciones UDG "A, B, C, D, E, F, G, H, I".

Ahora puedes darte cuenta de la ventaja que supone el tener una dirección de comienzo de zona de almacenamiento del *sprite,* como la F000h. Cada *sprite* comienza 72 bytes después de la dirección donde comenzó el anterior, lo que se corresponde con el valor "48h". Por tanto, los bytes menos significativos (LSB's) de las direcciones de comienzo serán 0, 72, 144 y 216 (00, 48, 90 y D8 en hexadecimal). Estos son todos menores de 256, por lo que no se alterarán los bytes más significativos (MSB's) de las direcciones.

A continuación encontrarás un programa de demostración en BASIC que hará todo lo visto en el párrafo anterior: "ad1" es el LSB de la dirección de la que hemos estado hablando y "ad2" es el MSB.

```
 170 PRINT AT 20,0;"Cualquier te
cla mostrara una secuencia anima
da": PAUSE 0
 180 CLS
 190 PRINT AT 20,0;"""O"" aborta
do"
 200 LET ad1=0: LET ad2=240
 210 FOR j=0 TO 3
 220 POKE 23675,ad1: POKE 23676,
ad2
 230 PRINT AT 10,14;"ABC";AT 11
,14;"DEF";AT 12,14;"GHI"
 240 PAUSE 5: LET AD1=AD1+72
 250 IF INKEY$="O" THEN  STOP
 260 NEXT J
 270 GO TO 200
```

Este programa ha sido realizado en forma de bucle, controlado por la variable "j". Sin embargo, cuando lo utilices en un programa (un juego o algo parecido) tendrías que hacer "POKE ad1" con los valores adecuados y dibujar el *sprite* en la pantalla, en cuatro puntos distintos de tu programa, es decir, distribuir los momentos de realizar las impresiones en pantalla entre distintos puntos del lazo principal de tu juego. En este caso, deberías despreciar la instrucción "PAUSE 5" de la línea 24, ya que un programa de juegos proporciona probablemente un retardo más que suficiente hasta la próxima ejecución de esta rutina.

Puedes imprimir el bloque de caracteres gráficos en cualquier lugar de la pantalla en que desees hacerlo y puedes organizar todo fácilmente para que la posición de impresión sea función de un control manejado por el jugador.

Para mostrarte lo que puedes hacer, en la figura 8.1 he dibujado un ciclo de animación que representa un hombre corriendo. Lo he realizado en una hoja cuadriculada de papel, donde cada cuadrado del dibujo se corresponde con un espacio de 8 × 8 bits o, lo que es lo mismo, el espacio de un carácter (véase próximo capítulo).

Observa que el cuerpo del hombre se desplaza hacia adelante un espacio equivalente a un cuarto de una posición de impresión en cada trama; esto significa que, al final de un ciclo completo (compuesto de cuatro tramas), puede comenzar a moverse de nuevo, pero ahora con el dibujo adelantado en una posición de impresión con respecto a cuando comenzó el anterior ciclo. El pie que está apoyado en la parte inferior de la hoja permanece en la misma posición mientras el dibujo se desplaza hacia adelante, pero el otro pie se mueve hacia adelante a lo largo del ciclo hasta bajar en una determinada posición.

Si realizas los dibujos para crear este movimiento y los introduces en un programa en BASIC, con un lazo del tipo FOR...NEXT, el cual mueva la posición de impresión un lugar a la izquierda en cada pasada del lazo, te sorprenderás de los resultados.

No te desilusiones si las figuras no son completamente iguales o si cometes algunos errores. Estos errores no se notarán cuando comience la acción; los movimientos extraños dan frecuentemente un carácter particular al dibujo.

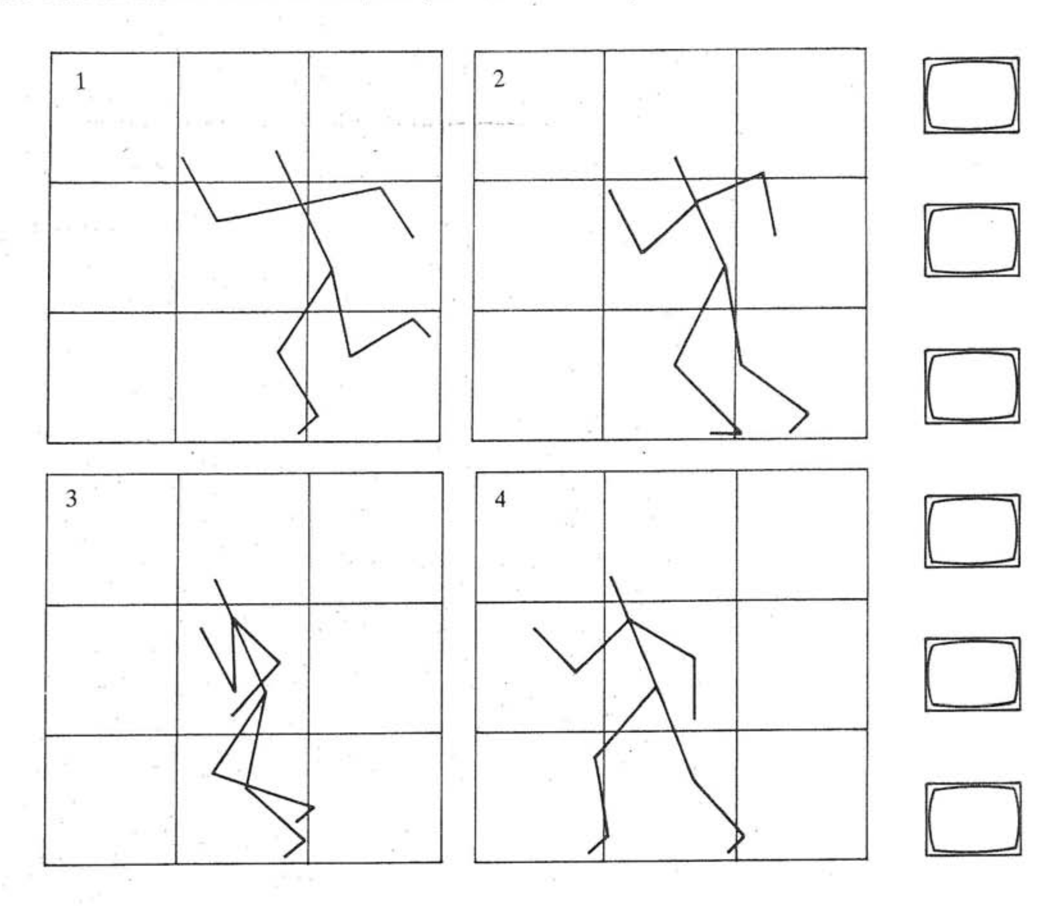
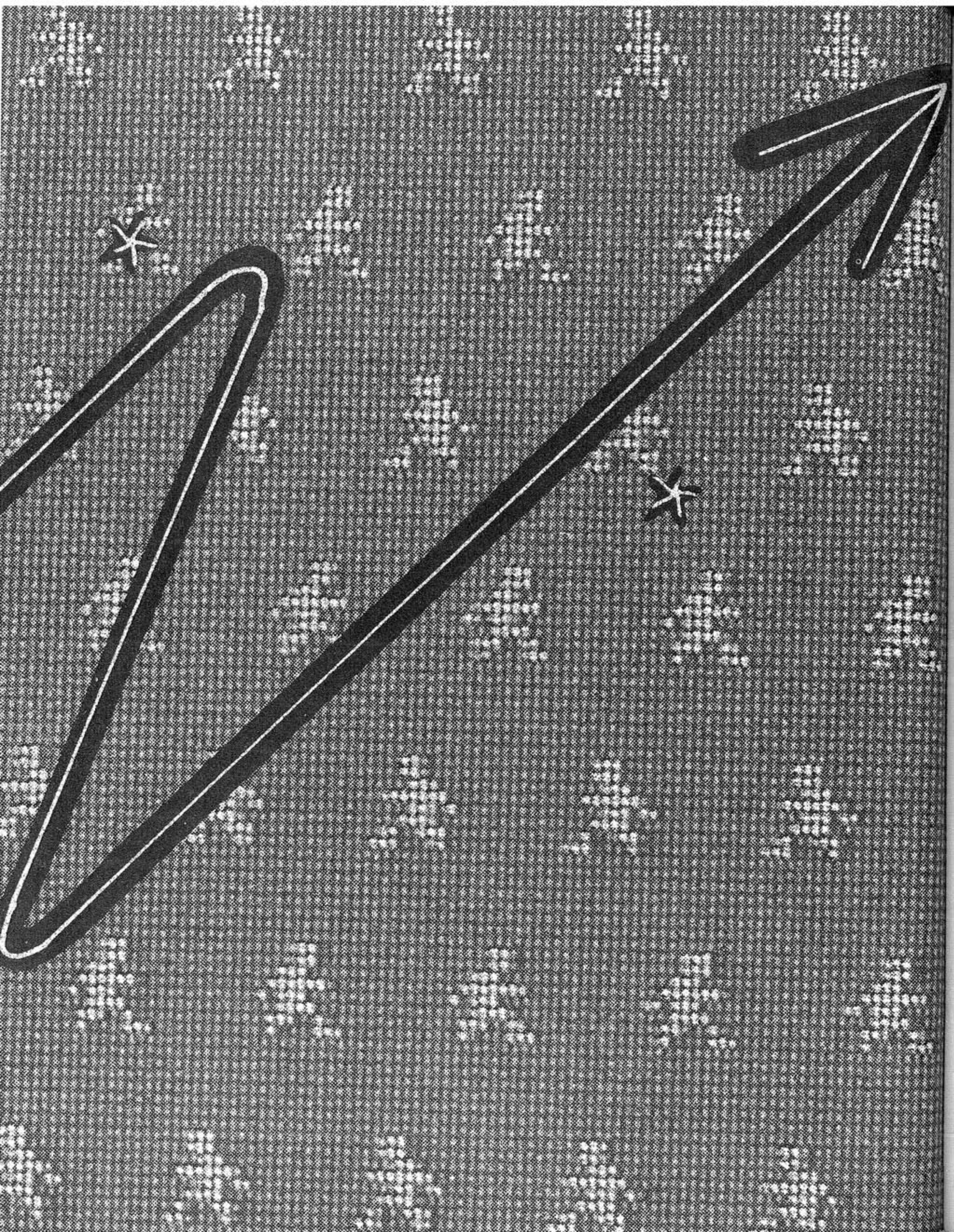



*Fig. 8.1*



# 9
# El movimiento del "sprite"

Cuando quieres generar el movimiento en el interior de un *sprite,* generalmente puedes realizar la mayor parte del programa necesario en BASIC. Utilizando la posición de impresión como una unidad de movimiento, el resultado es normalmente satisfactorio (tal y como se vio en el transcurso del capítulo anterior).

Sin embargo, cuando quieras mover un *sprite* como conjunto, por la pantalla, las cosas son un poco distintas. En el movimiento interno del *sprite* lo que se hace es un muy ligero movimiento de éste, de un punto de impresión cada vez.

## Organización del fichero de imagen

Usando el lenguaje de programación BASIC no hay forma de que puedas imprimir en la pantalla cosas que no se refieran a posiciones completas de impresión (8 × 8 bits). Pero para mover un *sprite* has de ser capaz de comenzar a dibujar en cualquier línea y columna de puntos de imagen (línea de *pixels*). Por tanto, has de olvidarte, para la realización de estos trabajos, de utilizar las instrucciones de impresión que posee el Spectrum, y trabajar directamente sobre los bytes del fichero de imagen.

Como podrás comprobar a continuación, esto no va a resultarte muy difícil.

He leído muchos artículos acerca de la organización del fichero de imagen del Spectrum, pero creo que son algo difíciles de comprender. Por esto, a continuación te lo explico de una forma un tanto distinta a como lo hacen en los artículos antes mencionados, pero tan útil como cualquiera de ellas.

Imagínate que el fichero de imagen está formado por tres líneas muy largas, cada una formada por 256 caracteres. Cada una de estas líneas ocupará un tercio de la imagen en

pantalla, en su dimensión horizontal. Cada una de las líneas imaginarias comienza en una dirección distinta del fichero de imagen (4000h, 4800h, 5600h).

Cada uno de los caracteres que forman las líneas está formado por ocho bytes (es decir, que cada línea imaginaria estará formada por ocho líneas de imagen en la pantalla); además, todas estas líneas imaginarias se almacenarán en el fichero de imagen, una detrás de otra. Por tanto, un carácter que ocupe la posición tres (como el de la figura 9.1) estará formado por el byte que está en la dirección 0 + 3 (relativa a la dirección de comienzo de la línea imaginaria primera, en el fichero de imagen); el segundo byte que lo formará ocupará en el fichero de imagen la posición relativa 256 + 3; el siguiente estará en la 512 + 3, y así sucesivamente.

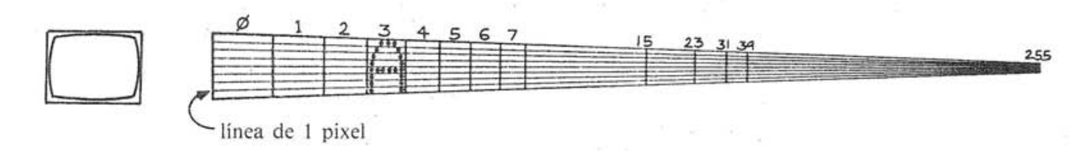



*Fig. 9.1. Representación de una de las tres líneas imaginarias de 256 caracteres*

Tal y como ya habrás pensado, las direcciones correspondientes a cada uno de esos ocho bytes que componen el carácter, serán del tipo siguiente: "X003h", "X103h", "X203h", etc. De hecho, para direccionar los ocho bytes que forman un carácter, con la dirección contenida en el registro HL, tendrás que incrementar dicho registro ocho veces (con "INC H") para poder acceder a los ocho bytes.

El formato de representación normal del Spectrum en pantalla está formado por líneas de 32 caracteres cada una. Para formar la pantalla con dicho formato, cada una de las líneas imaginarias del fichero de imagen (formada por ocho líneas de imagen cada una) será descompuesta en ocho líneas de caracteres en la pantalla, cada una con 32 caracteres (véase figura 9.2). Pero las direcciones de los bytes que forman esos caracteres siguen siendo las que antes explicamos para la línea imaginaria de 256 caracteres.

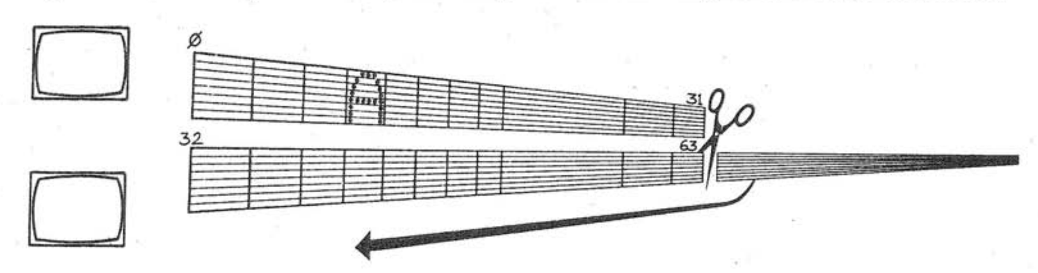



*Fig. 9.2. División de la línea imaginaria de 256 caracteres en ocho líneas de 32 caracteres*

Como habrás podido comprobar, en una visualización normal en pantalla encontrar la dirección de un carácter y escribir en cada uno de los bytes que la forman, y en la posición deseada, resulta bastante sencillo.

Pero supone que no queramos comenzar a imprimir un carácter a partir de la línea de imagen *(pixel line)* 0. ¿Qué ocurriría en este caso? Si, como hemos dicho, queremos

colocar el primer byte de nuestro carácter, por ejemplo, en la línea de imagen 4, ¿qué tendremos que hacer con el resto de bytes del carácter?

Los primeros cuatro bytes de este carácter pueden colocarse sin problemas en el espacio de la primera línea de caracteres de la pantalla utilizando el método antes descrito. Pero cuando lleguemos a la parte inferior de nuestra línea imaginaria del fichero de imagen y aún nos queden por colocar cuatro bytes del carácter que queremos introducir, ¿dónde los situaremos? Debe de haber algo debajo de esta línea de pantalla, a no ser que sea la más baja de la pantalla, pero ¿qué es lo que hay?

La respuesta es que, debido a la partición que se realiza de las líneas imaginarias, los bytes que ocupará lo que queda del carácter a introducir pertenecen a las líneas de imagen superiores del carácter situado en la pantalla debajo del ya ocupado, y que pertenecerá a la misma línea imaginaria del fichero de imagen, pero en una posición de "32d" (20h) caracteres más adelante (véase figura 9.3). Debido a esto, si estamos trabajando con la dirección del último byte de un carácter, la cual está almacenada en el registro DE, deberíamos disminuir esta dirección en un valor de "700h" (256 × 7) para regresar a la línea de imagen 0 (o primer byte de ese mismo carácter). Visto todo esto, podemos acceder al siguiente carácter a rellenar decrementando en uno el registro D y sumando "20h" al registro E (es decir, decrementando un valor de 256 × 7 en el registro D, con lo que estaremos en la línea de imagen 0 del carácter en el que estábamos y sumamos "20h" [32d] en el registro E, para posicionar la impresión en el carácter que está en la pantalla, justo debajo del que ya hemos escrito).

Si queremos pasar de una posición perteneciente a una línea imaginaria del fichero de imagen a una posición equivalente dentro de la siguiente línea imaginaria, tendremos que sumarle "0800h" (2048d) a la dirección que estemos usando en dicho momento.

Como puedes imaginarte, resulta complicado escribir un programa que haga todo esto, con bucles dentro de otros bucles (anidados), en el que cada uno debe trabajar con unos valores distintos, lo que obliga a realizar muchos almacenamientos y lecturas del *stack* (con PUSH y POP); y por si esto fuera poco, la dificultad queda aumentada en proporción al número de caracteres que forman el *sprite* a manejar.

Pero no te desanimes al pensar que tienes que resolver todo lo visto anteriormente, ya que Sinclair ha realizado todo este trabajo y ha incluido su resultado en la ROM del Spectrum. Hay una subrutina en la dirección 22AAh, llamada PIXEL_AD, que resuelve todas las preguntas que antes nos hemos planteado.

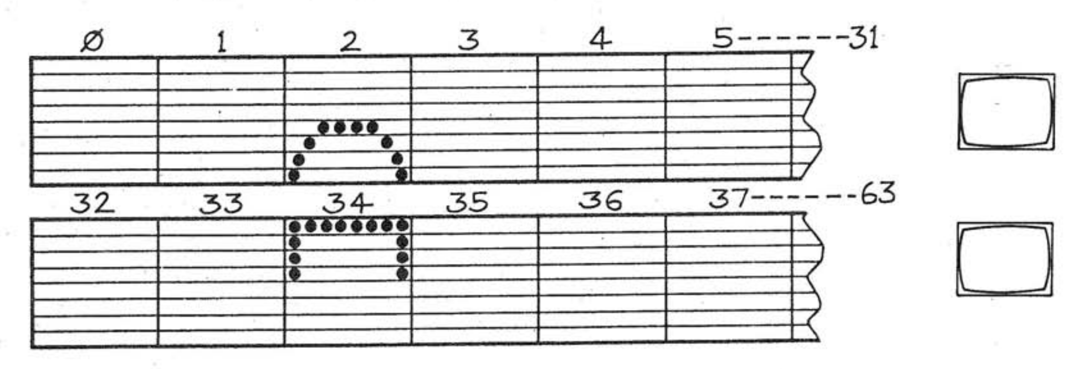



*Fig. 9.3. Impresión de un carácter entre dos líneas consecutivas*

Esta rutina se utiliza cuando se ejecuta la instrucción PLOT o se busca un punto de imagen con POINT. Nosotros colocamos en el par de registros BC las coordenadas X e Y de un determinado punto de imagen *(pixel);* hecho esto llamamos con CALL a la rutina "PIXEL_AD" y ésta nos devolverá en el registro HL la dirección en la cual se almacenará el byte al que pertenece el citado punto de imagen, para así colocarlo en el fichero de imagen en la posición adecuada. Pero además nos devolverá, en el registro A, la posición que ocupa el punto de imagen seleccionado, dentro del byte que hemos almacenado.

Según lo visto, el registro HL contendrá la línea de imagen sobre la que trabajamos (la coordenada Y), más el valor de la posición de impresión (ajuste grueso de la coordenada X), mientras que el registro A contendrá la posición exacta del punto de imagen sobre el que queremos trabajar (ajuste fino de la coordenada X).

Antes de pasar a ver cómo podemos utilizar las posibilidades que nos ofrece esta rutina en la práctica, puede que estés interesado en hacer un pequeño repaso de los conceptos sobre los que hemos trabajado anteriormente.

Cada una de las líneas imaginarias está formada por 256 caracteres, compuesto cada uno de ellos de ocho bytes. Como cada byte está formado por ocho puntos de imagen *(pixels),* se deduce que cada una de las líneas imaginarias estará formada por:

$$256 \times 8 \times 8 = 16.384 \textit{ pixels}$$

Como la pantalla está formada por tres de estas líneas imaginarias, nos resulta que la pantalla que utiliza el Spectrum está formada por:

$$3 \times 16.384 = 49.152 \textit{ pixels}$$

siendo cada uno de estos puntos de imagen direccionables por separado.

## Realización de un carácter desplazado verticalmente

Vamos a tratar primero el caso más sencillo. Suponte que imprimimos un carácter en la pantalla, utilizando una rutina que selecciona cualquier línea de puntos de imagen (coordenada Y) que deseemos.

El método a seguir consiste en almacenar las coordenadas de la esquina superior izquierda del carácter, en el registro BC, llamar a la rutina PIXEL_AD (CALL PIXEL_AD) y almacenar el primer byte del carácter en la dirección indicada por el registro HL. Hecho esto, has de pasar al próximo byte del carácter, decrementar la coordenada Y en uno (la coordenada Y se lee comenzando desde la parte más baja de la pantalla, hacia arriba) y volver a llamar a la rutina PIXEL_AD, antes de imprimir el próximo byte. Este proceso se repetirá hasta haberlo realizado con los ocho bytes del carácter.

Este tipo de trabajo es el idóneo para ser realizado con un bucle de control, para contar los ocho bytes, y el mejor de todos los bucles posibles en ensamblador es el realizado con la instrucción DJNZ. Esta instrucción necesita para su funcionamiento un número de control en el registro B, pero ya habíamos destinado anteriormente el registro BC para almacenar la dirección del elemento de imagen deseado. No podemos salvar dicho

registro en el *stack,* ya que es necesario que el contenido de este registro varíe en diferentes puntos del programa para que éste funcione correctamente. Para resolver este problema utilizaremos registros alternativos y accesos al registro BC.

El programa queda como sigue:

**Realización del carácter "A" a partir del punto de imagen con coordenadas (88, 172)**

```
F000                        ORG   #F000
F000 0158AC                 LD     BC,#AC58 ;Coordenadas
F003 ED5B7B5C               LD     DE,(#5C7B) ;Dir. variable UDG
F007 D9                     EXX
F008 0608                   LD     B,#08    ;Contador de un Byte
F00A D9                     EXX
F00B C5                     PUSH  BC
F00C CDAA22                 CALL  #22AA   ;Llama rutina PIXEL_AD
F00F 1A                     LD     A,(DE)
F010 77                     LD     (HL),A
F011 13                     INC   DE
F012 C1                     POP   BC
F013 05                     DEC   B
F014 D9                     EXX
F015 10F3                   DJNZ  #F00A
F017 D9                     EXX
F018 C9                     RET
```

Observa cómo el registro BC, en un principio, contiene el valor "AC58h" —ésta es la dirección hexadecimal que representan las coordenadas 172, 88—. En la línea "F003h", volvemos a tener el par de registros DE, direccionando a la variable del sistema UDG (5C7Bh), por lo que puedes cambiar la dirección de los gráficos alterando UDG y la rutina seguirá funcionando. Esto puede resultar útil para conseguir el efecto de animación.

A continuación encontrarás un programa sencillo en BASIC que te ayudará a comprender mejor el funcionamiento de la rutina:

**Programa de demostración**

```
  10 PRINT "XXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXX"
  20 RANDOMIZE USR 61440

XXXXXXXXXXXAXXXXXXXXXXXXXXXXXXXX
```

La línea formada por caracteres "X" te ayudará a poder situar el carácter gráfico "A" en cualquier posición de la línea.

## Realización de un carácter desplazado horizontalmente

Vamos ahora a considerar la forma en que podemos desplazar un carácter gráfico hacia los lados. Hasta ahora, sólo hemos realizado movimientos en sentido vertical. Este

nuevo movimiento nos lo hemos de plantear de una forma un poco diferente a como lo hicimos con el anterior.

Como te indiqué en el capítulo 7, no hay ninguna manera de poder desplazar directamente un carácter a la derecha o izquierda en el Spectrum.

Para realizar un carácter que esté desplazado lateralmente, hemos de realizar el carácter utilizando dos posiciones de impresión.

Como nos ocurrió anteriormente, tendremos que utilizar el bit de acarreo para contener los bits que quedan fuera de un byte al moverlo, y pasarlos después al byte adyacente del segundo carácter.

En cada rotación del byte, el carácter se moverá un bit (equivalente a un punto de imagen); por tanto, para poder llegar a la posición que deseemos hemos de utilizar el valor que queda almacenado en el registro A, al realizar la llamada a la subrutina "CALL PIXEL_AD", para conocer la posición exacta del punto de imagen y así poder controlar el número de rotaciones a realizar.

El código del programa es el siguiente:

**Desplazamiento de un carácter a la derecha, tantas veces como se indique en el registro A**

```
F020                ORG   #F020
F020 01AC5B         LD     BC,#5BAC ;Coordenadas
F023 CDAA22         CALL  #22AA   ;Llama PIXEL_AD
F026 4F             LD     C,A       ;Numero de desplazamientos
F027 A7             AND   A
F028 C8             RET   Z          ;Retorna si realizados
                 ;        todos los desplazamientos
F029 2158FF         LD     HL,#FF58 ;UDG "A"
F02C 1160FF         LD     DE,#FF60 ;UDG "B"
F02F 0608           LD     B,#08    ;Contador de un byte

F031 7E             LD     A,(HL)
F032 1F             RRA              ;Rotacion primer carac.
F033 77             LD     (HL),A
F034 1A             LD     A,(DE)
F035 1F             RRA              ;Rotacion segundo carac.
F036 12             LD     (DE),A
F037 23             INC   HL
F038 13             INC   DE
F039 10F6           DJNZ  #F031
F03B 0D             DEC   C
F03C 20EB           JR     NZ,#F029
F03E C9             RET
```

Como podrás observar, después de almacenar la dirección de comienzo en el registro BC, el programa llama a la rutina "PIXEL_AD" una vez y pasa el número que ésta devuelve en el registro A al registro C. Debido a este último cambio entre los registros, hemos perdido la información sobre la dirección de comienzo que teníamos en el registro BC, cosa que no nos importa en este momento, por lo que podemos reutilizar el registro BC para este otro propósito. Tampoco tenemos ningún interés en la nueva dirección contenida en el registro HL, por lo que utilizamos los registros HL y DE para almacenar en ellos las direcciones de los dos primeros gráficos definibles por el usuario, "A" y "B" (el Spectrum de 48K de memoria los tiene situados en las direcciones FF58h y FF60h).

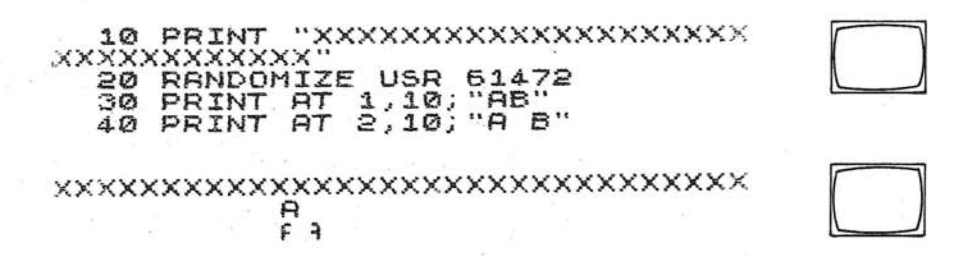
Las instrucciones "AND A" y "RET Z" son utilizadas para realizar controles de se-

guridad al programa. "AND A" chequea el registro A para ver si éste está a cero; si lo está, no necesitamos hacer ninguna rotación, por lo que salimos de la subrutina. Si no existiera esta verificación, la subrutina repetiría el ciclo 256 veces.

El bucle interno, controlado por el registro B, realiza la rotación de una pareja de bytes de los dos caracteres (tomando un byte de cada carácter) en cada pasada. El bucle exterior, controlado por el registro C, repite el bucle interno el número de veces que se indique en "C" (en este ejemplo son cuatro).

Antes de ejecutar el programa de demostración, debería poner los bytes de UDG "B" todos a cero, realizando almacenamientos de ceros en todos los bytes de dicho carácter (con POKE). Cuando hayas comprendido lo anterior, puedes ejecutar el siguiente programa:

**Programa de demostración para realizar la impresión de un carácter desplazado lateralmente**



```
   10 PRINT "XXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXX"
   20 RANDOMIZE USR 61472
   30 PRINT AT 1,10;"AB"
   40 PRINT AT 2,10;"A B"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
          A
          F 1
```

Como puedes ver, el carácter UDG "A" está impreso en una posición intermedia entre dos "X". De hecho, ya no es un solo carácter, sino que son dos caracteres los que forman la letra "A", tal y como puedes ver en la tercera línea impresa, en la que hay un espacio entre las dos mitades que forman dicho carácter.

## Realización de un "sprite'

Con los conocimientos adquiridos con la realización de las anteriores prácticas, estamos preparados para hacer frente a la realización de un *sprite* móvil. Pero antes de comenzar a analizar las rutinas, hay que realizar unos pequeños retoques al *sprite* para ponerlo a punto.

En primer lugar, realizaremos el *sprite* con una amplitud de *cuatro* bytes, en vez de tres; aun así, los gráficos seguirán siendo de 3 × 3. Esto se hace así para que con el byte que hemos añadido se pueda comenzar todo el proceso de movimiento horizontal, tal y como vimos en la anterior rutina.

En segundo lugar, todo resultará más sencillo si reorganizamos los bytes que forman el *sprite;* para esto podemos colocar todos los bytes que forman una línea horizontal, uno al lado del otro (tal y como está organizado el fichero de imagen en el Spectrum). La figura 9.4 muestra gráficamente lo que te estoy diciendo. Dicha figura muestra la forma en la que los gráficos están organizados en la parte alta de la RAM.

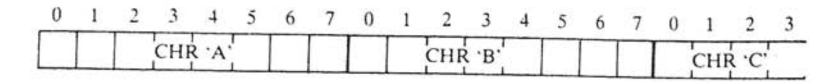



*Fig. 9.4*

También podemos reorganizarlo como el ejemplo de la figura 9.5.

| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A0 | B0 | C0 | 0 | A1 | B1 | C1 | 0 | A2 | B2 | C2 | 0 | A3 | B3 | C3 | 0 | A4 | B4 | C4 | 0 |

*Fig. 9.5*

Esta maniobra nos supone que podemos almacenar los bytes en el fichero de imagen de una forma fácil (con un movimiento sencillo), en vez de saltar ocho bytes en cada operación. Mejor aún, podemos realizar las operaciones de rotación ejecutando una línea completa cada vez.

La rutina que efectúa todas estas combinaciones es bastante sencilla, aunque aparecerá una cierta complicación después de efectuar los octavos movimientos, cuando ya hemos completado un grupo de caracteres y tenemos que moverlo hacia el siguiente grupo de caracteres.

Tendrás que elegir la dirección en la que copiar los datos con esta nueva organización, y te vuelvo a sugerir que utilices para ello el *buffer* de la impresora, que se encuentra situado debajo de la zona con las variables del sistema. Este espacio estará libre cuando lo necesitamos; es estable, no se ve afectado por el uso del *microdrive,* etc.

**Reorganización de los nueve primeros caracteres UDG en PR_BUFF**

```
F100            ORG   #F100
F100 21005B     LD     HL,#5B00 ;Buffer de impresora
F103 E5         PUSH  HL       ;Salva Dir. en Stack

F104 AF         XOR   A        ;Limpia el buffer de
F105 47         LD     B,A      ; la impresora
F106 77         LD     (HL),A
F107 23         INC   HL
F108 10FC       DJNZ  #F106

F10A D1         POP   DE       ;Recupera Dir.
F10B 2A7B5C     LD     HL,(#5C7B) ;Dir. de variable UDG
F10E 0603       LD     B,#03    ;3*Grupos de carac.
F110 C5         PUSH  BC
F111 0E08       LD     C,#08    ;8*Bytes por carac.
F113 0603       LD     B,#03    ;3*Carac. consecutivos
F115 E5         PUSH  HL

F116 7E         LD     A,(HL) ;Lazo para los bytes
F117 12         LD     (DE),A
F118 C5         PUSH  BC
F119 010800     LD     BC,#0008
F11C 09         ADD   HL,BC
F11D C1         POP   BC
```

```
F11E 13          INC   DE
F11F 10F5        DJNZ  #F116

F121 13          INC   DE
F122 E1          POP   HL
F123 23          INC   HL       ;Seleciona prox. carac.
F124 0D          DEC   C
F125 20EC        JR    NZ,#F113 ;Lazo para carac.
F127 0E10        LD    C,#10    ;"B"esta ya a 0
F129 09          ADD   HL,BC    ;Selecciona prox. grupo
F12A C1          POP   BC
F12B 10E3        DJNZ  #F110    ;Lazo para grupo
F12D C9          RET
```

La rutina comienza limpiando el *buffer* de la impresora (colocando ceros en todo este espacio), ya que deseamos tener un espacio limpio. Después inicializa los tres bucles, a tres, ocho y tres, respectivamente, y los ejecuta, incrementando la dirección original y la dirección de destino en cada pasada, y añadiendo dieciséis a la dirección original cuando ha finalizado con un conjunto de tres caracteres (añade dieciséis debido a que ya está en la posición 8, el final del carácter, y que por tanto sólo es necesario saltar sobre dos caracteres completos para comenzar con el cuarto).

A continuación puedes analizar el contenido de los primeros 64 bytes del *buffer* de la impresora. Los valores de los bytes no son importantes, pero puedes ver de una forma clara cómo están organizados en grupos de cuatro, con un blanco al final de cada grupo.

```
5B00 00 00 00 00 3C 7C 3C 00
5B08 42 42 42 00 42 7C 40 00
5B10 7E 42 40 00 42 42 42 00
5B18 42 7C 3C 00 00 00 00 00
5B20 00 00 00 00 78 7E 7E 00
5B28 44 40 40 00 42 7C 7C 00
5B30 42 40 40 00 44 40 40 00
5B38 78 7E 40 00 00 00 00 00
```

Después de estos pasos preliminares, el siguiente paso será la rotación de los bytes en nuestro espacio de memoria, de manera que desplacemos los caracteres a la posición que deseemos. Continuando con nuestro listado, éste tiene el aspecto siguiente:

**Desplazamiento de un "sprite" a la derecha tantas veces como se indica en el registro A**

```
F12D             ORG   #F12D
F12D 01AC5B      LD    BC,#5BAC ;Coord. punto imagen
F130 C5          PUSH  BC       ;Salva las coordenadas
F131 11005B      LD    DE,#5B00 ;Dir. buffer impresora
F134 CDAA22      CALL  #22AA
F137 4F          LD    C,A
F138 A7          AND   A
F139 280D        JR    Z,#F148 ;Salta si contador=0
F13B D5          PUSH  DE
```

```
F13C 0660        LD    B,#60    ;Rotacion de todos los
F13E 1A          LD    A,(DE)   ; caracteres del buffer
F13F 1F          RRA            ; de impresora
F140 12          LD    (DE),A
F141 13          INC   DE
F142 10FA        DJNZ  #F13E

F144 D1          POP   DE
F145 0D          DEC   C
F146 20F3        JR    NZ,#F13B ;Repite la operacion
F148 C1          POP   BC       ;Recuper coordenadas
F149 C9          RET
```

Podrás observar que este listado es muy parecido al que utilizamos anteriormente en este capítulo para desplazar el carácter a la derecha. Es bastante más sencillo, ya que los bytes de los caracteres han sido reorganizados en una forma más accesible.

Cuando el programa se está ejecutando, la misma zona de memoria que anteriormente observaste en el listado de memoria, ahora tomará el siguiente aspecto:

```
5B00 00 00 00 00 03 C7 C3 C0
5B08 04 24 24 20 04 27 C4 00
5B10 07 E4 24 00 04 24 24 20
5B18 04 27 C3 C0 00 00 00 00
5B20 00 00 00 00 07 87 E7 E0
5B28 04 44 04 00 04 27 C7 C0
5B30 04 24 04 00 04 44 04 00
5B38 07 87 E4 00 00 00 00 00
```

Como las coordenadas que hemos elegido (que fueron almacenadas en el registro BC en la dirección de programa F12Dh) requieren un desplazamiento de cuatro posiciones de imagen, observarás que los mismos números en hexadecimal aparecen ahora claramente desplazados al interior del siguiente espacio de impresión.

Lo siguiente que hemos de hacer es colocar el *sprite* ya desplazado en el fichero de imagen situándolo en la línea de imagen horizontal, de la forma que hemos especificado.

La rutina vuelve a ser muy parecida a la que desplazaba el carácter a la derecha.

### Impresión, en la pantalla, de un "sprite"

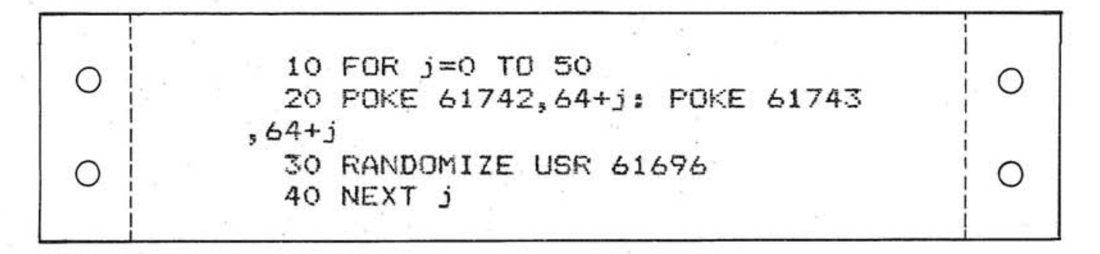
```
F149             ORG   #F149
F149 DD21005B    LD    IX,#5B00 ;Dir. buffer impresora
F14D D9          EXX
F14E 0618        LD    B,#18    ;Contador de 3*8 lineas
                                  de puntos de imagen
F150 D9          EXX
F151 C5          PUSH  BC
F152 CDAA22      CALL  #22AA    ;Llama PIXEL_AD
F155 0604        LD    B,#04    ;Contador anchura Sprite
F157 DD7E00      LD    A,(IX)
F15A 77          LD    (HL),A
F15B 23          INC   HL
F15C DD23        INC   IX
```

```
F15E 10F7        130         DJNZ  #F157
F160 C1          140         POP   BC
F161 05          150         DEC   B
F162 D9          160         EXX
F163 10EB        170         DJNZ  #F150
F165 D9          180         EXX
F166 C9          190         RET
```

Notarás que he usado el registro índice IX para almacenar la dirección de la zona de memoria utilizada para los *sprites,* en vez del registro DE. Esto se debe a que la capacidad de realizar direccionamientos indexados será utilizada en capítulos siguientes (véase rutina para generación de caracteres híbridos, en el capítulo 10).

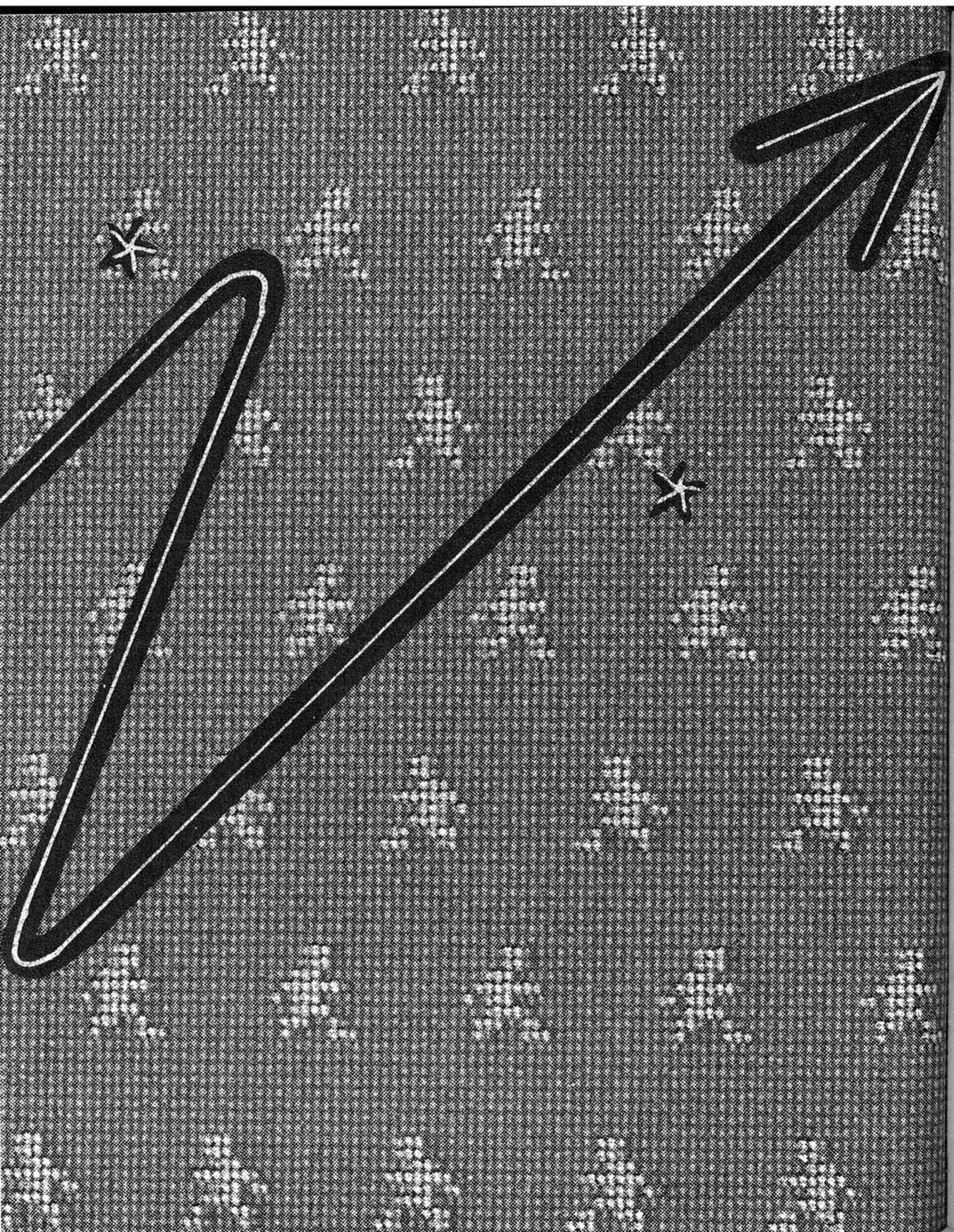
Mientras, si pasas a memoria un programa en BASIC como este:

**"Sprite" móvil**



```
  10 FOR j=0 TO 50
  20 POKE 61742,64+j: POKE 61743
,64+j
  30 RANDOMIZE USR 61696
  40 NEXT j
```

puedes hacer que tu *sprite* se mueva por toda la pantalla, como una nube en el cielo. Esto es posible de realizar tal cual, si dibujas una nube y la introduces en los caracteres UDG, tal y como hemos explicado anteriormente.



# 10

# Realización de los fondos sobre los que se mueven los "sprites"

Si has colocado tu *sprite* en la pantalla, puede ser que hayas observado un inconveniente de las rutinas anteriores tal y como las hemos desarrollado hasta ahora. Si hay otro dibujo en la pantalla, éste es borrado por el *sprite,* el cual siempre aparecerá en el interior de un rectángulo blanco, tal y como se ve en la figura 10.1.

*Fig. 10.1*

Además, si mueves el *sprite,* sobre el dibujo aparece un efecto parecido al de una aspiradora, destruyendo de esta manera todos los dibujos que forman el fondo de la pantalla (véase figura 10.2).

## Protección del dibujo de fondo de la pantalla

No resulta muy difícil la restauración del dibujo de fondo que ha resultado destruido al mover el *sprite*. Esto se resuelve si almacenas una copia del fichero de imagen, que contenga tu dibujo de fondo, en una zona adecuada de la memoria RAM, y la recuperas cada vez que colocas tu *sprite* en una nueva posición de la pantalla. Esta técnica de volcado de la pantalla es otra herramienta muy útil del código máquina, la cual puede utilizarse para un buen número de aplicaciones, aunque esto significa la pérdida de una zona de RAM al utilizarla para contener una copia del fichero de imagen.

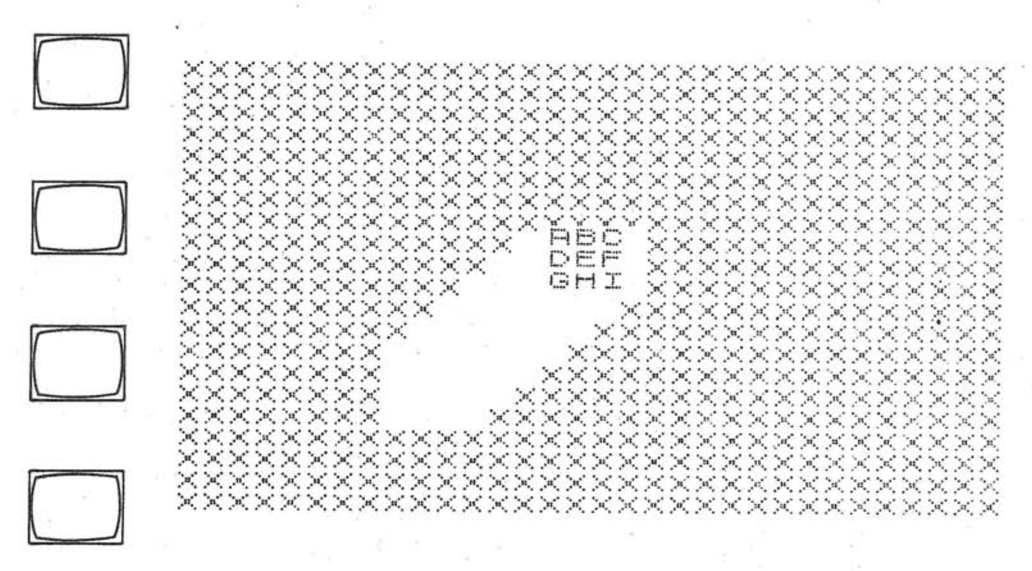



*Fig. 10.2*

Esta codificación es difícilmente mejorable: se trata de una operación LDIR.

### Volcado de la pantalla

```
F000                ORG   #F000
F000 210040         LD     HL,#4000
F003 1100D0         LD     DE,#D000
F006 01001B         LD     BC,#1B00
F009 EDB0           LDIR
F00B C9             RET
```

Esta rutina almacena el fichero de imagen en la zona de memoria comprendida entre la dirección "D000h" y la "EAFFh". Para recuperar esta copia en la pantalla, el programa será más o menos el mismo.

**Recuperación del volcado de la pantalla**

```
F010              10        ORG   #F010
F010 2100D0       20        LD     HL,#D000
F013 110040       30        LD     DE,#4000
F016 010016       40        LD     BC,#1600
F019 EDB0         50        LDIR
F01B C9           60        RET
```

Las rutinas para el volcado y recuperación de la pantalla se ejecutan de una forma tan rápida que son casi instantáneas. En realidad, puedes utilizarlas para hacer aparecer y desaparecer una línea de texto o un diseño (esto se hace de una forma sencilla almacenando el dibujo de fondo, volcándolo, y luego, de una forma alternativa, escribir el texto y restaurar la imagen).

## "Sprites" con capacidad para moverse libremente

Después de lo realizado anteriormente, aún seguirás teniendo el *sprite* dentro de su cuadrado blanco. El efecto conseguido hará que en vez de tener una nave espacial, por ejemplo, sobre la pantalla, en realidad tengas una especie de postal de una nave espacial, lo cual de alguna manera resulta menos atrayente.

Para conseguir que el *sprite* se mueva libremente sobre cualquier fondo, has de encontrar la manera de dibujar un fondo que rodee todos los bordes del *sprite,* no importando cómo esté construido el *sprite* o en qué parte del rectángulo, en el que está incluido, se encuentra.

Esto no resulta en la realidad tan difícil como pueda parecer al exponer el problema. De hecho, para resolver este problema vamos a utilizar una técnica que es tan antigua como lo es la industria del cine, que tiene aproximadamente ochenta años de antigüedad.

Permíteme que te cuente algo acerca de la primera película provista con movimientos en las escenas. Fue realizada en 1903 y su título era "El gran robo al tren". Su duración es de once minutos. La historia es bastante sencilla: hay un tren que transporta una gran cantidad de lingotes de oro; los ladrones se introducen en el interior del vagón donde estaba guardado el oro; todo esto se produce mientras el tren se mantiene en movimiento; por último, y como es de esperar, los ladrones roban el oro. La escena más espectacular se produce cuando los ladrones reducen al guarda y escapan con el botín; al mismo tiempo que se producen estas escenas, se pueden observar las vistas de una ciudad a través de una puerta abierta del vagón.

Lo que yo quiero ahora utilizar es el hecho de que no había tal ciudad cuando se realizó el film; para conseguir el efecto deseado, se colocó en la puerta una pieza de cartón pintada de blanco.

Después de esto, la película se volvió a rodar desde un tren en movimiento, a través

de una pieza de cartón negro con un agujero, acoplando de una forma exacta la puerta blanca antes citada, con el agujero realizado en el cartón negro. Esto supuso el invento de la impresión sobre transparencias.

Esta técnica, u otras como ésta, aún se utiliza en la televisión, donde se denomina *Chromakey*. Esta técnica es muy utilizada en las noticias dadas por televisión, en las que los locutores pueden aparecer sobre un fondo de explosiones de bombas, etc., mientras permanecen leyendo en su silla. Normalmente procurarás evitar el utilizar esta técnica, debido a que aparece una línea alrededor de los dibujos que están en primer plano, muy parecida a la que producen los rotuladores de punta gruesa (aspecto de una línea con pelos).

Para realizar un proceso como el visto, de impresión sobre transparencias, por cualquier técnica, es necesario utilizar cuatro elementos básicos. Dichos elementos se muestran en la figura 10.3.

La figura "A" representa el fondo, la "C" es el primer plano. La figura "B" se denomina "transparencia positiva" y la "D" es denominada "transparencia negativa".

Para realizar la figura compuesta en la pantalla, en primer lugar se combinan las figuras "A" y "B", utilizando la "B" para enmascarar el espacio para el primer plano. Hecho esto, la figura "C" es introducida en el espacio en blanco creado por la aplicación de la transparencia positiva, utilizando la transparencia negativa "D" para enmascarar la zona dedicada al fondo. Todo esto se ve en la figura 10.4.

En las películas, esto se realiza combinando de una forma física las partes de un film en una impresora óptica; pero, para conseguir el mismo efecto en un ordenador, podemos utilizar las instrucciones lógicas del procesador central para enmascarar adecuadamente los bits deseados y añadir otros en su lugar.

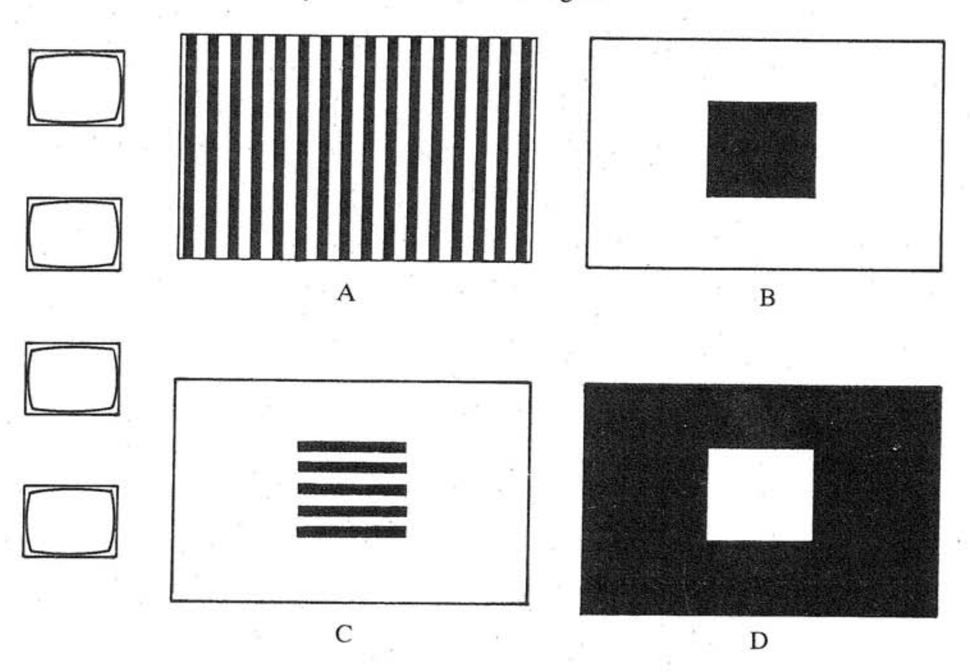



*Fig. 10.3. Cuatro elementos del proceso de transparencia*

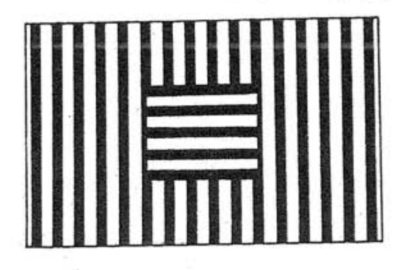


*Fig. 10.4. Figura compuesta*

A continuación puedes analizar una rutina que genera un carácter "híbrido" en el espacio de los gráficos redefinibles por el usuario. La rutina toma los primeros caracteres UDG, "A" y "B" y utiliza una transparencia adecuada realizada en el UDG "C", y coloca el carácter híbrido final en el carácter UDG "D".

**Generación de caracteres híbridos**

```
D000                ORG   #D000
D000 DD2A7B5C       LD     IX,(#5C7B) ;Dir. variable UDG
D004 0608           LD     B,#08
D006 DD7E10         LD     A,(IX+#10) ;Caracter "C"
D009 2F             CPL
D00A DDA600         AND   (IX)      ;Caracter "A"
D00D 4F             LD     C,A
D00E DD7E10         LD     A,(IX+#10) ;Caracter "C"
D011 DDA608         AND   (IX+#08) ;Caracter "B"
D014 B1             OR     C
D015 DD7718         LD     (IX+#18),A ;Caracter "D"
D018 DD23           INC   IX        ;Incrementa Dir. base
D01A 10EA           DJNZ  #D006
D01C C9             RET
```

```
  10 LET n=0
  20 FOR j=0 TO 7: POKE USR "C"+
j,15: NEXT j
  30 RANDOMIZE USR 53248
  40 PRINT TAB 10;"A B C D"
```

En la línea 20 del programa en BASIC, utilizado para la demostración anterior, se genera el efecto de transparencias. Los caracteres compuestos aparecen de una forma más clara en la segunda línea obtenida como resultado de la demostración, donde los caracteres rayados han sustituido a los anteriores UDG "A" y "B".

Resulta muy interesante analizar la forma en que trabaja la rutina. En la posición D000h se almacena la dirección de comienzo del UDG en el registro IX. En D004h se

crea un bucle de control de ocho pasadas, correspondientes a los ocho bytes que forman cada carácter y que son los que tendremos que utilizar para trabajar con ellos. En la posición D006h se carga el registro A con el primer byte procedente del tercer carácter, el carácter con la transparencia. La siguiente instrucción crea el carácter denominado "transparencia negativa"; la instrucción CPL invertirá todos los bits del byte almacenado en el registro A. En D00Ah se borran los bits que forman la parte central del carácter UDG "A", al realizar la operación AND entre los dos bytes (el contenido en el registro A y cada uno de los ocho que forman el UDG "A"). En D00Dh se almacena el resultado de la anterior operación en el registro C. En la posición D00Eh se vuelve a cargar la transparencia por segunda vez y en D012h se utiliza como transparencia positiva para enmascarar, en el carácter UDG "B", la parte que anteriormente no se borró en el UDG "A"; dejando ahora la ventana central del carácter libre, para colocar en ella lo que se quiera. En D014h se combinan las dos partes creadas anteriormente (utilizando la operación OR) y en D015h se carga el nuevo byte en el carácter UDG "D". Después de todo esto, incrementamos la dirección contenida en el registro IX para que apunte al siguiente byte y volvemos a realizar las mismas operaciones anteriores otras siete veces más.

Puedes comprobar en esta rutina la ventaja que supone el poder utilizar la posibilidad que nos ofrece el registro de indexación IX. Esto te permitirá direccionar, respecto a la misma posición relativa, a los cuatro caracteres que tenemos que manejar en la rutina. Como la rutina trabaja con los ocho bytes de cada carácter, sólo tendrás que incrementar el valor de la base (dirección contenida en el registro IX).

Utilizaremos la misma técnica para direccionar al *sprite* y a su "transparencia" asociada.

## Impresión de "sprites" sobre un fondo

Hacer que la rutina trabaje con bloques de 12 bytes supone realizar unos pequeños ajustes en sus bucles de control. Para asegurarnos de que siempre accederemos a un fondo de imagen limpio (es decir, a un lugar sin imágenes anteriores impresas en él) realizaremos una copia, antes de ejecutar la rutina, de todo el fichero de imagen en una zona de la RAM. Utilizaremos esta copia como base para la generación del fondo para el *sprite*. La copia del fichero de imagen se puede realizar con la rutina para el volcado de la pantalla, que yo he colocado en la dirección F180h. Esta rutina carga una copia del fichero de imagen en un bloque compuesto de 1800h bytes, que comienza en la dirección D800h y finaliza un byte antes de la F000h.

Siempre podemos encontrar en la copia del fichero de imagen la dirección que ocupa el *sprite* en el fichero de imagen actual, añadiendo un desplazamiento constante de 9800h a la dirección del fichero de imagen. Este desplazamiento es almacenado en el registro DE, en la primera instrucción de la rutina siguiente:

**Impresión de un "sprite" sobre un fondo**

```
F149                ORG   #F149
F149 110098         LD     DE,#9800 ;Desplazamiento para
                              hacer copia fichero
                              de imagen
F14C DD21005B       LD     IX,#5B00
F150 D9             EXX
F151 0618           LD     B,#18
F153 D9             EXX
F154 C5             PUSH  BC
F155 CDAA22         CALL  #22AA
F158 0604           LD     B,#04
F15A E5             PUSH  HL
F15B 19             ADD   HL,DE    ;Toma Dir. copia fichero
                            de imagen
F15C DD7E60         LD     A,(IX+#60) ;Trasparencia negativa
                               en "A"
F15F 2F             CPL        ;Hace la traspar. negativa
F160 A6             AND   (HL)     ;Enmascara el fondo
F161 4F             LD     C,A
F162 DD7E60         LD     A,(IX+#60) ;Transp. negativa en "A"
F165 DDA600         AND   (IX)     ;Enmascara el Sprite
F168 B1             OR     C        ;Forma imagen compuesta
F169 E1             POP   HL       ;Recupera Dir. fichero
                            de imagen original
F16A 77             LD     (HL),A
F16B 23             INC   HL
F16C DD23           INC   IX
F16E 10EA           DJNZ  #F15A
F170 C1             POP   BC
F171 05             DEC   B
F172 D9             EXX
F173 10DE           DJNZ  #F153
F175 D9             EXX
F176 C9             RET
```

**Volcado del fichero de imagen a la dirección D800h (51200d)**

```
F180                ORG   #F180
F180 210040         LD     HL,#4000
F183 1100D8         LD     DE,#D800
F186 010018         LD     BC,#1800
F189 EDB0           LDIR
F18B C9             RET
```

Lo más probable es que, al analizar las anteriores rutinas, te preguntes qué es lo que significa el desplazamiento "IX + 60", que se realiza en la rutina de impresión. Este desplazamiento nos situará en la dirección de la transparencia. Los caracteres UDG que forman el *sprite* son los siguientes: A B C D E F G H I, y los que forman la transparencia son: J K L M N O P Q R. Cada uno de estos grupos utiliza 60h (96d) bytes. Esto supone realizar dos pequeñas modificaciones en las rutinas que hemos desarrollado anteriormente:

F10E LD B,03 06 03 se convertirá en: LD B,06 06 06

y

F13C LD B,60 06 60 se convertirá en: LD B,C0 06 C0

Como verás, ambos números se han duplicado.

Colocando todo junto, en la figura 10.5 puedes observar un programa de demostración y su resultado.

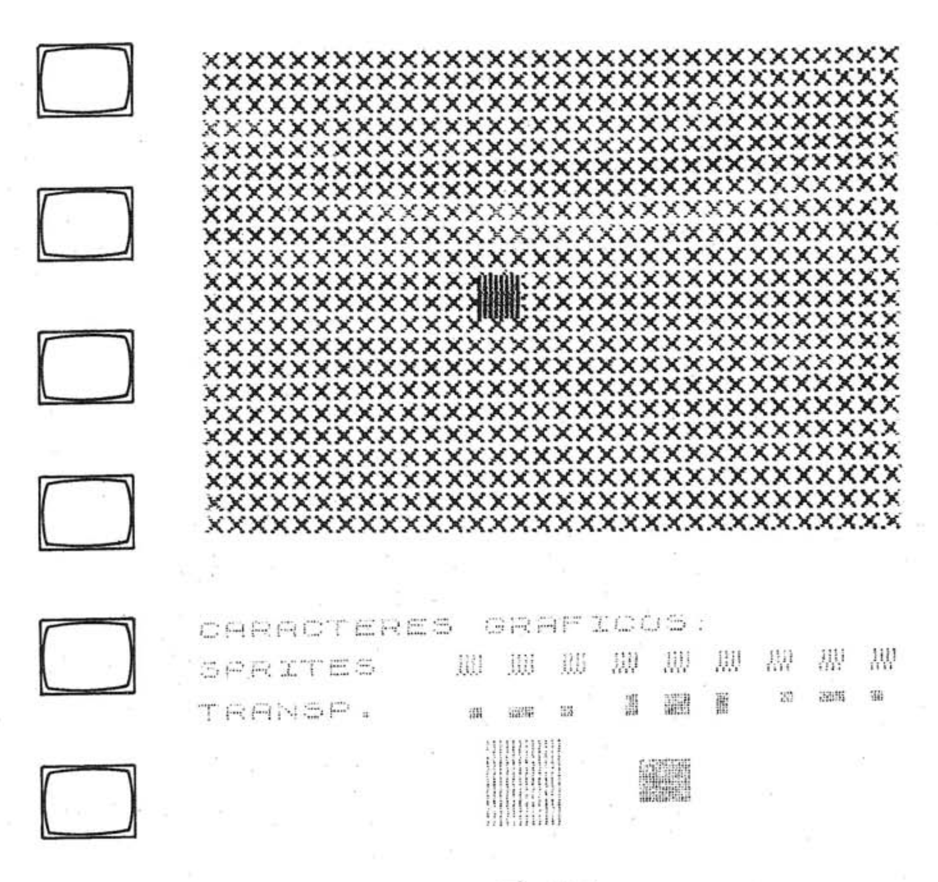



*Fig. 10.5*

**Programa de demostración sobre la realización de un "sprite" con las técnicas de transparencias**

```
 10 FOR j=0 TO 703: PRINT "X";:
NEXT j
 20 RANDOMIZE USR 61824
 30 POKE 61742,97: POKE 61743,9
7
 40 RANDOMIZE USR 61696
```

El *sprite* puede no ser compacto; como en los realizados hasta ahora, podemos hacer que el centro del cuadrado que forma el *sprite* sea transparente, dando como resultado la figura 10.6.

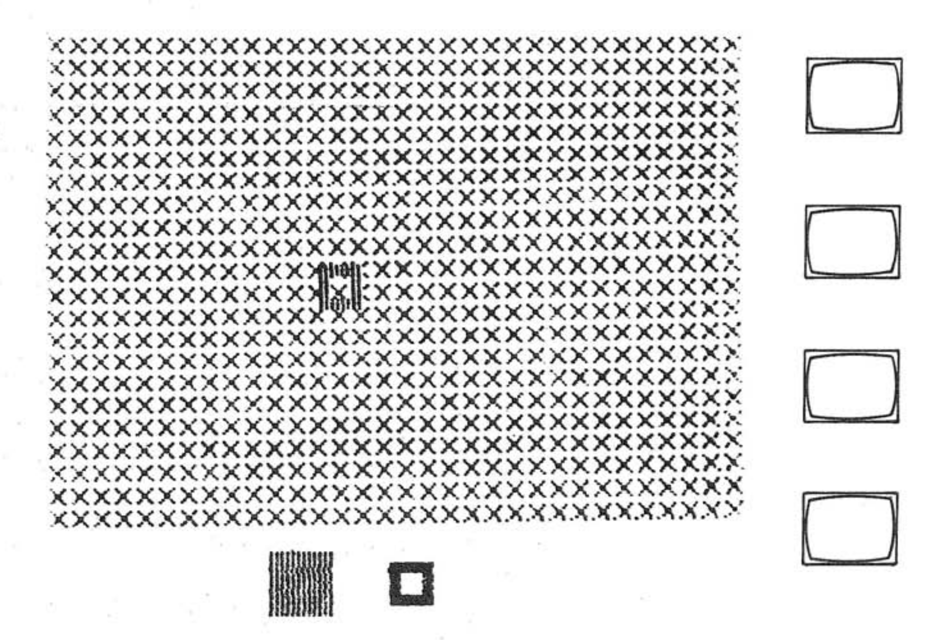
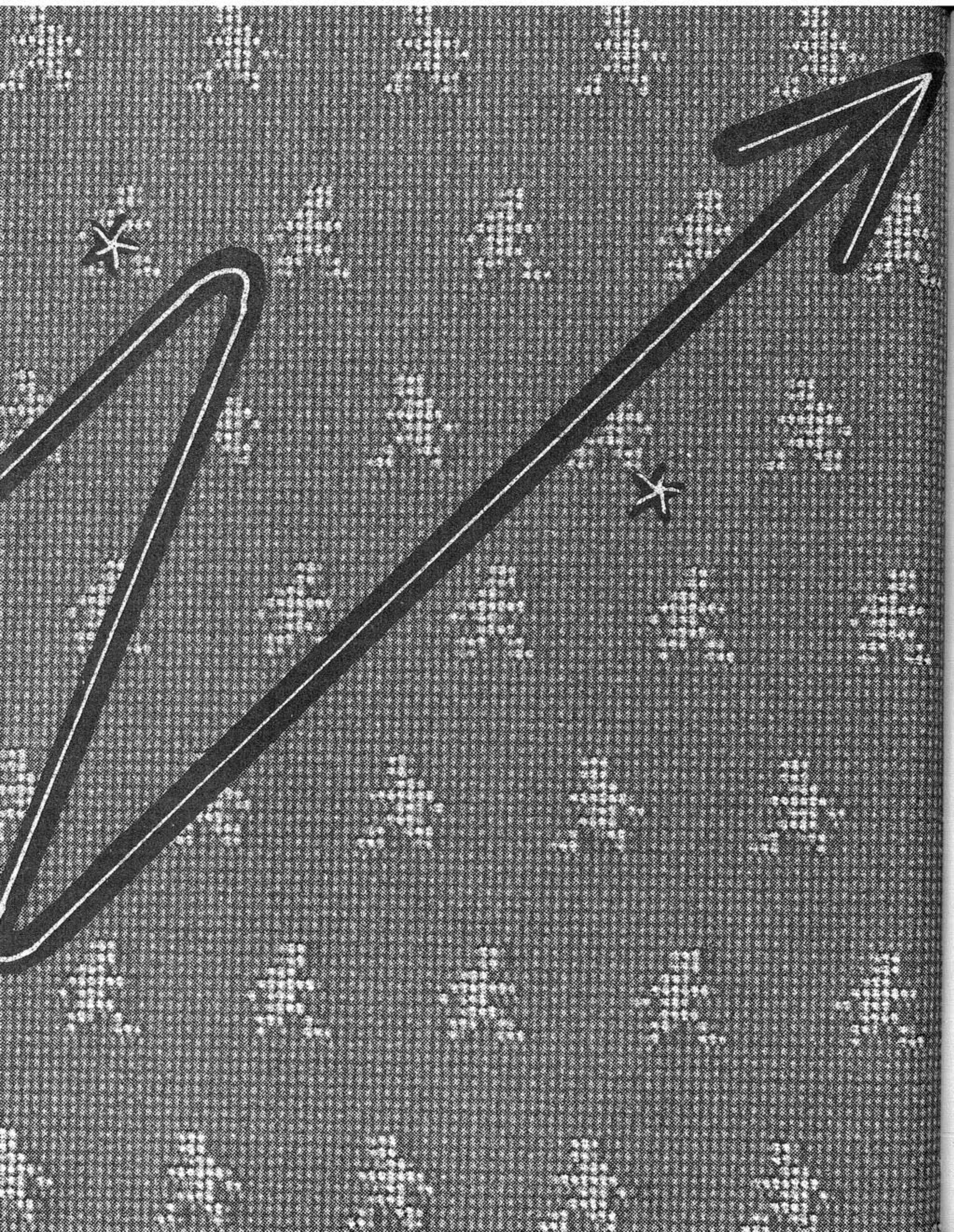


*Fig. 10.6*

Todo lo visto hasta ahora no supone que hayas agotado todas las posibilidades de manejo de tu *sprite.* Copiando todas las rutinas en dos posiciones de memoria diferentes, y asignando otra zona de memoria RAM para que contenga una copia del fichero de imagen, puedes tener dos *sprites* funcionando al mismo tiempo.

Si utilizas una copia del fichero de imagen con el *sprite* 1 ya colocado en el fichero, de forma que éste sea un fondo transparente para el *sprite* 2, puedes hacer que el *sprite* 2 pase por delante del *sprite* 1.

Puedes realizar más efectos utilizando la técnica de las transparencias. Realizando una transparencia en una zona de la imagen de fondo (puedes hacerlo utilizando la rutina "PaintCODE" que se da en el Apéndice A), puedes hacer que un *sprite* pase detrás de un edificio. Lógicamente, esto implica la utilización de una gran zona de memoria, pero el fichero de imagen es fácilmente descompuesto en tres partes, con lo que puedes utilizar y almacenar solamente una parte de las tres que lo forman.



# 11

# El fichero de atributos

En el Spectrum, el fichero de imagen y el de atributos están relacionados el uno con el otro tan estrechamente, que prácticamente no te enteras de que se trata de dos cosas distintas y que se utilizan para unas aplicaciones totalmente distintas. Tú ya sabes que puedes imprimir en la pantalla en varios colores, sobre unos determinados colores de fondo, únicamente utilizando los distintos campos que te ofrece la instrucción PRINT.

Sin embargo, incluso con las operaciones más sencillas, no podemos apreciar cómo trabajan los dos ficheros. Separando las funciones de ambos, puedes aumentar las posibilidades de operación de una forma apreciable.

El fichero de atributos es, en realidad, un fichero de imagen con una resolución más baja que el fichero de imagen que utiliza el Spectrum. Este fichero está compuesto por tan sólo 768d bytes de información, frente a los 6144d bytes que forman el fichero de imagen propiamente dicho; pudiendo, por tanto, trabajar el fichero de imagen con 49.152 puntos de imagen distintos.

Esta comparación de datos te permitirá comprender el porqué no se pueden asignar colores distintos a cada punto de imagen por separado en el Spectrum. Si cada punto de imagen tuviera un byte de atributos, necesitaríamos utilizar unos 60K de RAM para cada línea del programa. Esto supone que con el Spectrum no podrás realizar una imagen en la que una pequeña flecha roja se desplace sobre una raya formada con puntos de imagen. Incluso en el Sinclair QL, el cual posee una memoria más amplia, cada punto de la imagen sólo tiene cuatro bits de atributos.

El fichero de atributos está organizado de una forma bastante clara. Observando el fichero de atributos como un fichero aislado, puedes utilizarlo como método para tomar decisiones, controlando la aparición (desaparición) de caracteres de la pantalla. Esto

proporciona, por ejemplo, un método excelente de saber qué es lo que ocurre en distintos puntos de la pantalla, sin que sea necesario que se aprecie a simple vista.

El siguiente programa es una muestra de lo que te ha explicado anteriormente.

**Atributos no visibles de una posición de la pantalla**

```
  10 DIM l$(32)
  20 LET x=INT (RND*32)
  30 PRINT  PAPER 0; INK 7;l$
  40 PRINT AT 0,x; PAPER 0; INK
5;" "
  99 STOP
 100 FOR j=0 TO 31: IF PEEK (225
28+j)<>7 THEN  PRINT j
 110 NEXT j
 199 STOP
 200 PRINT AT 0,0
 210 FOR j=0 TO 31: PRINT  OVER
1; INK 8;"A";: NEXT j
```

Las líneas de la 10 a la 30 crean una línea negra, con una letra "X" que se posiciona en lugares aleatorios de dicha línea. No hay ninguna forma de hacer visible la "X" en la pantalla; para los ojos todas las posiciones de la línea son iguales. Pero, aunque invisible a los ojos, en el fichero de atributos se puede apreciar una diferencia entre la posición ocupada por "X" y el resto de las posiciones de la línea; la posición en la que está la "X" tiene el atributo INK a un valor de 5, mientras que los valores del mismo atributo, para el resto de las posiciones de la línea, es 7 en todas ellas. Esta diferencia se mantiene, aunque no haya que imprimir nada con el atributo INK.

Pulsando CONTINUE comienza una búsqueda del byte diferente y se imprimirá el número en el cual se encuentra dicho byte. Pulsando CONTINUE por segunda vez, aparecerá la posición oculta de una forma visual; se presentará en pantalla una línea compuesta de letras "A", todas en blanco, excepto la que ocupe la posición buscada, que aparecerá en otro color.

Esta rutina para "esconder y buscar" puede resultarte muy útil para realizar pequeños juegos. Como ejemplo, el siguiente programa genera una baraja de cartas con las que trabaja en la pantalla de una forma invisible para nosotros (es el efecto de que las cartas estuvieran boca abajo). A partir de la línea 100, comienza a volver las cartas. Estas aparecen en la pantalla de color blanco, con las figuras y los valores en el color correspondiente. Dejo a tu imaginación la búsqueda de un método para barajar las cartas antes de utilizarlas y para representar los dibujos de las figuras y el tablero de juego. También tendrás que decidir qué es lo que haces con las dos cartas que no tienen espacio para aparecer en la pantalla.

```
   9 LET b=0: LET k=1: LET s=0
  10 FOR i=0 TO 4: FOR j=1 TO 30
 STEP 3
  20 LET x$=CHR$ (s+144): LET b=
NOT b: LET x=1+12*(k-1)
  30 IF k>9 THEN  GO TO 200
  40 LET q$=x$+"          "+x$+x
$+"   "+x$+"      "+x$+x$+" "+x$
+"      "+x$+" "+x$+x$+" "+x$+"
"+x$+"    "+x$+" "+x$+x$+" "+x$+
x$+" "+x$+"   "+x$+" "+x$+x$+" "
+x$+x$+x$+x$+"   "+x$+" "+x$+x$+
" "+x$+x$+" "+x$+x$+" "+x$+x$+"
"+x$+x$+" "+x$+x$+x$+x$+x$+" "+x
$+x$+" "+x$+x$+" "+x$+x$+x$+x$+x
$+x$+x$+x$+" "+x$
  50 PRINT  PAPER 3; INK 3; BRIG
HT b; OVER 1;AT i*4+1,j;q$(x TO
x+2);AT i*4+2,j;q$(x+3 TO x+5);A
T i*4+3,j;q$(x+6 TO x+8);AT i*4+
4,j;q$(x+9 TO x+11)
  60 LET k=k+1: IF k>13 THEN  LE
T k=1: LET s=s+1
  70 NEXT j
  80 LET b=NOT b
  90 NEXT i
 100 PAUSE 0
 105 LET k=1: LET c=0
 110 FOR i=0 TO 4: FOR j=1 TO 30
 STEP 3
 120 PRINT  PAPER 7; INK (2 AND
c); BRIGHT 8; OVER 1;AT i*4+1,j;
"   ";AT i*4+2,j;"   ";AT i*4+3,
j;"   ";AT i*4+4,j;"   "
 130 LET k=k+1: IF k>13 THEN  LE
T k=1: LET c=NOT c
 150 NEXT j: NEXT i
 199 STOP
 200 IF k=10 THEN  LET q$=x$+"
 J  J    "+x$
 210 IF k=11 THEN  LET q$=x$+"
 Q  Q    "+x$
 220 IF k=12 THEN  LET q$=x$+"
```

```
K  K    "+x$
230 IF k=13 THEN  LET q$=x$+"
A  A    "+x$
240 LET x=1: GO TO 50
```

Puedes observar el uso que se hace del atributo BRIGHT (brillo) para diferenciar los contornos de las cartas que están juntas, incluso en el caso de que ambas fueran del mismo color. El atributo BRIGHT resulta muy útil, ya que te proporcionará la posibilidad de tener seis y posiblemente siete nuevos tonos de color (el negro no se ve afectado por este atributo y la diferencia, en el caso del color azul, es casi inapreciable), todo dependerá de tu televisor.

La posición de los puntos se almacena en la variable q$, mientras que la variable X$ se fija a uno de los palos de la baraja, según el valor de la variable "s", el cual es añadido a la posición del primer carácter UDG (144).

Desgraciadamente, no es posible mostrar en el libro una representación de la pantalla completa, ya que la impresora ZX ignora cualquier atributo, por lo que no se puede reflejar el resultado de esta rutina con claridad.

El fichero de atributos se encuentra colocado de una forma compacta entre las posiciones de memoria 5800h y la 5B00h.

El primer byte del fichero guarda los atributos de la posición de impresión que ocupa la esquina superior izquierda de la pantalla y, a partir de esta posición, se colocan todas las posiciones de la pantalla ordenadamente hasta la esquina inferior derecha, que estará 300h (768d) posiciones después del primer byte de este fichero.

Debido a que este fichero está colocado en la memoria de una forma sencilla (al contrario que el fichero de imagen), resulta muy fácil calcular tu posición en la pantalla y acceder directamente al fichero de atributos (con POKE).

También puedes leer posiciones de dicho fichero (con PEEK) para controlar la posición que está ocupda y la que no lo está. Esta es una manera de superar el problema que supone el hecho de que la opción SCREEN$ no trabaja con los gráficos definibles por el usuario. Si tus caracteres gráficos utilizan unos atributos INK o PAPER particulares (o unos BRIGHT o FLASH particulares) puedes comprobar, dentro de la pantalla, en qué situación se encuentran, mirando en las distintas posiciones del fichero (con PEEK) o utilizando directamente la opción ATTR, con lo que calcularás la dirección en que está tu carácter.

Por último, esta disposición lógica del fichero de atributos hace que la operación de realizar dibujos en él sea más fácil. El programa siguiente realiza todas estas operaciones:

**Dibujos utilizando la opción ATTR**

```
10 LET x=16: LET y=10
20 PAUSE 0: LET y$=INKEY$
30 IF CODE y$<48 THEN  GO TO 5
```

```
40 LET c=CODE y$-48
50 IF CODE y$=8 THEN  LET x=x-
(x>0)
51 IF CODE y$=9 THEN  LET x=x+
(x<31)
52 IF CODE y$=10 THEN  LET y=y
+(y<23)
53 IF CODE y$=11 THEN  LET y=y
-(y>0)
60 IF CODE y$=13 THEN  GO TO 1
00
70 POKE 22528+x+y*32,c*8
80 GO TO 20
```

En la línea 10, las variables X e Y fijan una dirección inicial en el centro de la pantalla y dibujan un cuadrado de color. Las condiciones "X > 0", "X < 31", etc., se utilizan para limitar el desplazamiento del cuadrado antes dibujado por la pantalla. Estas condiciones trabajan como funciones lógicas en las que el resultado es 1 si la condición se cumple o 0 si dicha condición es falsa. Por tanto, se añadirá un 1 a las variables X o Y solamente en el caso de que dichas variables definan una posición que se encuentre dentro del límite de la pantalla (tal y como se describió en el capítulo 8). Este programa te permitirá acceder a cualquier posición de las 24 líneas de la pantalla.

El programa fija un determinado color en la línea 30, después de lo cual podrás mover el cuadrado utilizando las teclas de movimiento del cursor (CAPS SHIFT con las teclas 5, 6, 7 u 8).

No he incluido en este listado el código necesario para poder realizar desplazamientos en diagonal del cuadrado, pero puedes encontrar un programa parecido en el apéndice A (el denominado "PaintCODE"), que te enseñará cómo puedes realizar estos movimientos.

El resultado es parecido al que se obtendría con una pluma electrónica y resulta ideal para realizar dibujos de fondos para escenas en la pantalla o de cualquier otra cosa que desees dibujar. Recuerda que, como sólo has trabajado con los atributos, los caracteres impresos normalmente pueden ser compilados y visualizados sobre el fondo de forma independiente, aunque has de incluir las opciones "PAPER 8; INK 9" para asegurarte de que éstos no introducen sus propios atributos en la imagen.

Tu impresionante obra maestra necesita ser almacenada en algún lugar, si no quieres que se pierda en cuanto realices un listado de tu programa (LIST). Puedes realizar esto utilizando la instrucción SAVE...SCREEN$, pero esta instrucción tarda bastante tiempo en efectuar el almacenamiento y la posterior carga, además de que en su ejecución almacenará el contenido del fichero de imagen, cosa que tú no deseas realizar en este momento.

Resulta más rápido y claro copiar el fichero de atributos en un lugar de la RAM y almacenarlo como un bloque de datos con la instrucción "SAVE ATTR CODE xxxxx,768". Esta instrucción tarda menos en ejecutarse que la anterior y coloca la

copia del fichero en un lugar en el que pueda ser recuperado durante la ejecución de tu programa.

Para realizar una copia del fichero, puedes utilizar un programa como el siguiente:

**Almacenamiento del fichero de atributos en la dirección F000h (61440d)**

```
10 FOR j=0 TO 768
20 POKE 61440+j,PEEK (22528+j)
30 NEXT j
```

Un programa en código máquina resulta incluso más sencillo. Realiza su trabajo de una forma más rápida; tarda alrededor de 10 segundos; también realiza el trabajo inverso de colocar de nuevo tu diseño en la pantalla, siempre que lo desees.

**Almacenamiento del fichero de atributos en la dirección F000h**

```
F500                  ORG    #F500
F500 210058           LD      HL,#5800
F503 1100F0           LD      DE,#F000
F506 010003           LD      BC,#0300
F509 C9               RET
```

**Recarga del fichero de atributos desde la dirección F000h**

```
F50C                  ORG    #F50C
F50C 210050           LD      HL,#5000
F50F 1100F0           LD      DE,#F000
F512 010003           LD      BC,#0300
F515 EDB0             LDIR
F517 C9               RET
```

Las dos rutinas son llamadas utilizando el formato "USR 62720" y "USR 62732", siempre que mantengas las direcciones que aparecen en los anteriores listados.

Los 768d bytes utilizados para almacenar el fichero de atributos no ocupan un gran espacio extra de RAM, especialmente si lo comparas con los 6144d que ocupa el fichero de imagen. Sin embargo, ambos almacenarán una gran cantidad de información redundante para nuestros objetivos, ya que nosotros sólo estamos interesados en los colores de fondo (PAPER).

Es posible extraer la información sobre el color de fondo y almacenarla en tan sólo 288d bytes.

## El color

El sistema de almacenamiento de los bits que contienen la información sobre el color nos introducirá en el ámbito general de la reproducción en color. La fotografía, la impresión sobre papel y las imágenes creadas por un ordenador utilizan los mismos principios básicos para realizar sus trabajos con el color. Esto no te resultará sorprendente si piensas que la observación del color depende de la ausencia o presencia de tres colores básicos, lo cual es la base de los sistemas de reproducción del color que anteriormente he mencionado.

Vamos a abordar los principios generales en primer lugar.

El Spectrum guarda la información referente al color del fondo en los bits 3, 4 y 5 de cada byte del fichero de atributos. El primero de estos bits nos indica el estado del color azul (presente o ausente), el segundo el del color rojo y el tercero el del color verde. Suponiendo que estos tres bits no se encontraran, como realmente lo están, desplazados tres posiciones hacia la izquierda, los tres colores primarios están codificados por los estados de los bits 1 (azul), 2 (rojo) y 3 (verde), equivalente a los valores 1, 2 y 4, respectivamente. Si ahora observas en el teclado del Spectrum el número de la tecla correspondiente a cada color, verás que todo lo anterior resulta sencillo a la hora de saber el número de cada color.

Los otros tres colores que el Spectrum puede realizar resultan de la combinación de dos de los colores primarios, por lo que son llamados colores complementarios; estos colores son el "cyan" (nombre procedente del tinte que se utilizaba originalmente en fotografía para crearlo), el magenta y el amarillo. El blanco resulta de la combinación de los tres colores primarios. Los valores numéricos de cada color son los que se corresponden con sus respectivas teclas en el Spectrum.

Los colores complementarios son también conocidos como "colores inferiores", ya que el magenta, por ejemplo, es el resultado de quitar al blanco el color verde, el amarillo es blanco menos azul y el "cyan" es blanco menos rojo.

Te preguntarás qué tiene que ver todo esto con el Spectrum; pues bien, para darte una idea, suponte que extraemos todos los bits que ocupan la posición 3 dentro de los bytes que forman el fichero de atributos y los almacenamos; al hacer esto, habríamos almacenado la componente azul de la imagen que hay en la pantalla. Los bits que ocupan la posición 4 hacen lo mismo con la componente roja y los bits que ocupan la quinta posición lo hacen con la componente verde de la imagen. La información sobre los colores complementarios sería almacenada si almacenáramos pares de bits, de cada byte del fichero de atributos; la información sobre el blanco se conseguiría almacenando los tres bits de información de color que hay en los citados bytes.

Para volver a colocar estos bits en el fichero de atributos, de forma que la pantalla recobre sus colores originales, podemos añadir los colores primarios a una pantalla en negro, es decir, sin ninguna imagen (en términos de ordenadores, a esto se le denomina fijación de los bits primarios en un byte en blanco), o podemos sustraer los colores complementarios de una pantalla en blanco, lo que significa que pondremos a cero los pares de bits de colores complementarios, en unos bytes en los que inicialmente estaban a uno.

Todas estas operaciones anteriores se pueden realizar utilizando las operaciones lógicas de que dispone la pastilla del microprocesador Z80. A continuación vamos a ver algunos programas para realizar dichas operaciones. Como podrás observar, la parte más

complicada del trabajo es la de colocar los bits y bytes en el orden correcto y en la posición correcta.

La primera rutina resulta un poco más complicada de lo normal, ya que he colocado los bits con la información sobre el color de fondo (''PAPER colour''), los cuales extraigo y posteriormente almaceno, para poder organizarlos como caracteres que puedo luego imprimir como UDG's.

Represento toda esta información sobre la pantalla en un bloque de 4 × 3 caracteres.

El control lo realizo chequeando dónde está a uno el bit 3 de cada byte del fichero de atributos. Si lo está, pongo a uno un bit del registro A y traslado de una forma eventual el byte que contiene este registro a una posición de memoria. Mientras tanto, he desplazado todo el fichero de atributos un bit hacia la derecha, con lo que el bit que originalmente ocupaba la posición 4 ahora pasa a ocupar la posición 3. Cuando realizo la operación de nuevo, los bits con la información sobre el color rojo ya habrán sido almacenados. Al final de la segunda pasada, el fichero de atributos se volverá a desplazar un bit hacia la derecha, con lo que el bit que nos informa del color verde ocupará la posición 3.

Al final de la rutina se vuelve a colocar en su estado original todo el fichero de atributos.

**Separación de los tres colores del fichero de atributos**

```
F000           10       ORG   #F000
F000 1100F1    20       LD     DE,#F100 ;Dir. comienzo almacenaje
F003 0603      30       LD     B,#03    ;Contador para tres pasadas
               40 ;               para el Azul,Rojo y Verde
F005 C5        50       PUSH  BC
F006 210058    60       LD     HL,#5800 ;Comienzo Atributos
F009 0E03      70       LD     C,#03    ;Vertical(3 poiciones)
F00B 0608      80       LD     B,#08    ;Vertical(8 lineas por
               90 ;               posicion de impresion
F00D C5       100       PUSH  BC
F00E D5       110       PUSH  DE
F00F 0E04     120       LD     C,#04    ;Horizontal(4 posiciones)
F011 0608     130       LD     B,#08    ;Horizontal(8 bits por
              140 ;               posicion de impresion
F013 AF       150       XOR   A
F014 CB5E     160       BIT   3,(HL)   ;Chequea bit 3 de Atributos
F016 2801     170       JR     Z,#F019 ;Salta si esta a 0
F018 37       180       SCF             ;Pone a 1 acarreo si
              190 ;              bit 3 esta a 1
F019 17       200       RLA             ;Pasa acarreo a A
F01A CB0E     210       RRC   (HL)     ;Desplaza los atributos
              220 ;               para poner el bit 4 en
              230 ;               la posicion del bit 3
F01C 23       240       INC   HL
F01D 10F5     250       DJNZ  #F014
              260
F01F 12       270       LD     (DE),A  ;Pasa A al byte de almacenamiento
F020 C5       280       PUSH  BC      ;Mueve la Dir. de almac.
F021 EB       290       EX     DE,HL   ; 8 bytes para apuntar al
F022 0E08     300       LD     C,#08   ; proximo UDG
F024 09       310       ADD   HL,BC
F025 EB       320       EX     DE,HL
F026 C1       330       POP   BC
F027 0D       340       DEC   C
```

```
F028 20E7      JR     NZ,#F011 ;Repite la operacion con
               ;          4 caracteres UDG
F02A D1        POP    DE
F02B 13        INC    DE        ;Apunta siguiente byte
F02C C1        POP    BC
F02D 10DE      DJNZ   #F00D  ;Repite las operaciones
               ;         para los 8bytes del UDG
F02F C5        PUSH   BC       ;Mueve Dir. almacenamiento
F030 EB        EX     DE,HL    ; al siguiente grupo de
F031 0E18      LD     C,#18    ; 4 caracteres
F033 09        ADD    HL,BC
F034 C1        POP    BC
F035 EB        EX     DE,HL
F036 0D        DEC    C
F037 20D2      JR     NZ,#F00B ;Repite para 3 grupos
F039 C1        POP    BC
F03A 10C9      DJNZ   #F005  ;Repite para Azul,Rojo Verde
F03C 010003    LD     BC,#0300 ;Restaura los atributos
F03F CB06      RLC    (HL)     ; desplazados a su estado
F041 CB06      RLC    (HL)     ; original
F043 CB06      RLC    (HL)
F045 2B        DEC    HL
F046 0B        DEC    BC
F047 78        LD     A,B
F048 B1        OR     C
F049 20F4      JR     NZ,#F03F
F04B C9        RET
```

Los tres bloques de caracteres deberán almacenarse en la memoria, comenzando en la posición F100h (61696d) para el azul, por la F160h (61792d) para el rojo y por la F1C0h (61888d) para el verde. La rutina se coloca a partir de la posición F000h (61440d).

A continuación encontrarás un pequeño programa en BASIC para ejecutar el código máquina y después visualizar las tres imágenes resultantes de la separación de los colores primarios. Para visualizar estas imágenes, el programa introduce las direcciones adecuadas en la variable del sistema UDG (con POKE), 5C7Bh y 5C7Ch (23675d y 23676d).

Colócalo con el programa en BASIC para "Dibujar con la opción ATTR".

**Generación y visualización de las imágenes resultantes de la separación de los colores primarios**

```
 100 RANDOMIZE USR 61440
 110 POKE 23675,0: POKE 23676,24
1
 120 PRINT  INK 8; INVERSE 1;"AB
CD"'"EFGH"'"IJKL"
 130 POKE 23675,96
 140 PRINT  INK 8; INVERSE 1;"AB
CD"'"EFGH"'"IJKL"
```

```
150 POKE 24675,192
160 PRINT  INK 8; INVERSE 1;"AB
CD"'"EFGH"'"IJKL"
```

Lo que aparece en la figura 11.1, con letras indicando el color en que están realizadas, muestra cómo se realiza la separación de los colores primarios. Cada muestra en miniatura, en el borde inferior izquierdo de la figura, representa un color complementario. El recuadro exterior, que está realizado en negro, aparecerá en cada una de las figuras resultantes de la separación de los colores primarios. Si pudieras realizar transparencias de cada una de las imágenes resultantes, y colocarlas unas sobre otras, tendrías como resultado la imagen original.

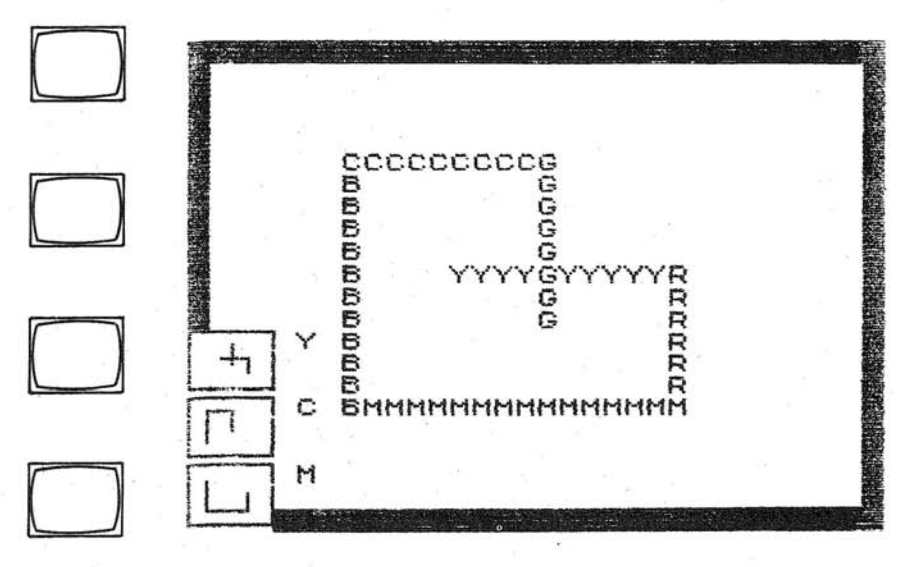



*Fig. 11.1*

A continuación hay una rutina para volver a reunir todos los atributos de nuevo.

**Reconstrucción del fichero de atributos**

```
F050                    ORG   #F050
F050 DD2160F1           LD     IX,#F160 ;Dir. imagen en Rojo
F054 210058             LD     HL,#5800 ;Fichero atributos

F057 0E03               LD     C,#03   ;Bucles control-1
F059 0608               LD     B,#08   ;2
F05B C5                 PUSH  BC
F05C DDE5               PUSH  IX
F05E 0E04               LD     C,#04   ;3
F060 0608               LD     B,#08   ;4
```

```
F062 DDCBA006    RLC   (IX-#60) ;Dir. de Azul
F066 3802        JR     C,#F06A
F068 CB9E        RES   3,(HL)  ;Pone a 0 bit Azul=Amarillo
F06A DDCB0006    RLC   (IX)     ;Dir. de Rojo
F06E 3802        JR     C,#F072
F070 CBA6        RES   4,(HL)  ;Bit Rojo a 0=imagen Cyan
F072 DDCB6006    RLC   (IX+#60) ;Dir. de Verde
F076 3802        JR     C,#F07A
F078 CBAE        RES   5,(HL)  ;Bit Verde a 0=Magenta
F07A 23          INC   HL
F07B 10E5        DJNZ  #F062  ;Bucle 4

F07D 110800      LD     DE,#0008
F080 DD19        ADD   IX,DE
F082 0D          DEC   C
F083 20DB        JR     NZ,#F060 ;Bucle 3

F085 DDE1        POP   IX
F087 DD23        INC   IX
F089 C1          POP   BC
F08A 10CF        DJNZ  #F05B  ;Bucle 2

F08C 1E18        LD     E,#18
F08E DD19        ADD   IX,DE
F090 0D          DEC   C
F091 20C6        JR     NZ,#F059 ;Bucle 1

F093 C9          RET
```

Como puedes ver, esta rutina trabaja bastante con el registro para direccionamientos indexados IX. Esto nos permite acceder a la información almacenada sobre el color, siguiendo un mismo método para todos los casos. Los desplazamientos necesarios para acceder a posiciones equivalentes en los tres distintos bloques de información, uno por cada color primario, serán siempre los mismos.

La dirección base para realizar la indexación[1] apuntará al bloque que almacenará la información referente al color verde. Este bloque es el que está en medio de los tres. Lo hago de esta manera porque el registro sólo puede realizar indexación de 128 bytes. Como cada color es almacenado en un bloque de 96 bytes, no sería posible realizar la indexación hasta el tercer bloque, si la base apuntara al primer bloque. Sin embargo, como la indexación puede realizarse aumentando 128 bytes a la base, o restándole 127 bytes a la misma, si la base apunta al bloque que está enmedio, podrás acceder por indexación a los bloques anterior y posterior.

Antes de finalizar el tema de los atributos, me gustaría comentar que las técnicas que hemos visto en este capítulo resultan muy útiles para crear pequeños dibujos para realizar los *sprites*. A mucha gente le resulta más fácil realizar los dibujos para el *sprite* ocupando toda la pantalla y, después de esto, reducir su tamaño hasta el que tendrá luego el *sprite*. Podrías también intentar utilizar la técnica de separación de los colores primarios, para crear un ciclo de tres representaciones para realizar el efecto de animación.

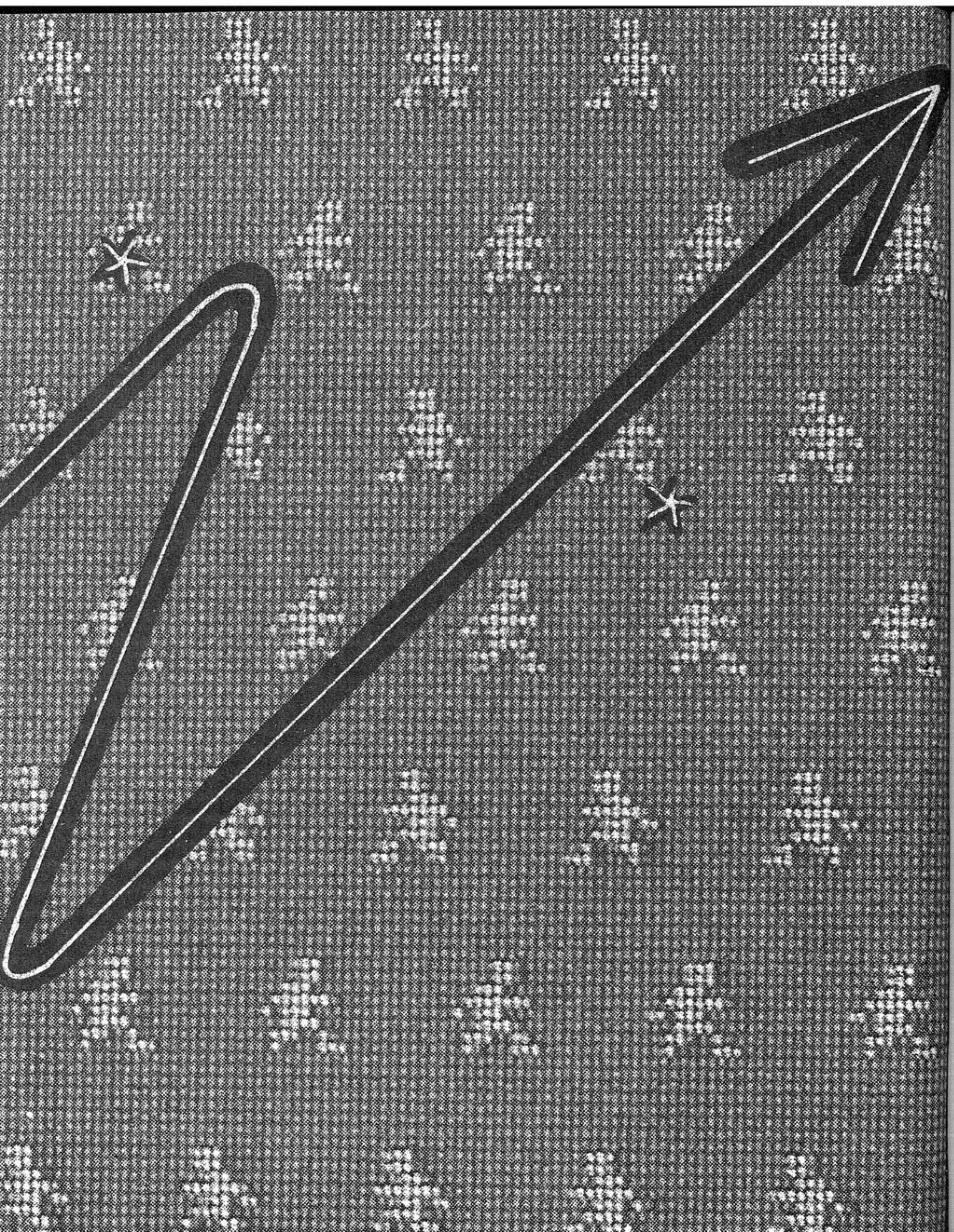
---

1 Indexación: Técnica utilizada en los direccionamientos. Esta técnica requiere el uso de dos cantidades, la base y el *Offset* (desplazamiento). La base es una dirección que se mantiene fija durante todo el proceso y es utilizada como dirección de referencia. El desplazamiento es la cantidad que puede variar y es el valor que se suma o se resta a la base para apuntar a la dirección deseada.

Si quieres intentarlo, a continuación encontrarás un programa en BASIC para cambiar los colores de los atributos de un dibujo de la pantalla.

**Puesta a verde o rojo de dibujos ya creados**

```
  99 REM Pone la imagen en rojo
 100 FOR j=0 TO 768: IF PEEK (22
528+j)<>7*8 THEN  POKE 22528+j,1
6
 110 NEXT j
 190 PAUSE 0
 199 REM Pone la imagen en verde
 200 FOR j=0 TO 768: IF PEEK (22
528+j)<>7*8 THEN  POKE 22528+j,3
2
 210 NEXT j
```



# 12
# El fichero de imagen

Mientras que el conocimiento del fichero de atributos resulta de interés, el fichero principal de imagen juega un papel mucho más importante en el trabajo normal realizado con el Spectrum. En una situación crítica, tú podrías trabajar sin el fichero de atributos, pero si lo intentaras sin el fichero de imagen el ordenador quedaría virtualmente ciego y mudo.

Antes de que empieces a manipular en el fichero principal de imagen del Sinclair, has de saber cómo trabaja y cómo está organizado. Como ya se ha apuntado en otros capítulos, no hay nada tan directo como el fichero de atributos. El fichero de imagen, aparte de ser mucho más grande que el de atributos, su organización interna está realizada de una forma no muy convencional (véase capítulo 9). Necesitas tener muy claras las ideas sobre esto, para saber el camino que sigues a través del fichero de imagen. No te debe preocupar el que el *Manual del Spectrum* recalque a menudo el hecho de que: "La organización del fichero es tan curiosa, que probablemente no te apetezca realizar operaciones PEEK o POKE en él. Lo más probable es que desees hacerlo."

Trabajando con los *sprites,* hemos utilizado hasta ahora, en bastantes ocasiones, la rutina residente en la ROM, PIXEL_AD. La utilizamos para conocer la dirección que ocupa una posición de la imagen, dentro del fichero de imagen, después de calcular las coordenadas X e Y de dicha posición en la pantalla. Pero si, por ejemplo, quieres manejar toda la imagen, punto a punto de imagen, esta rutina no te servirá para resolver este problema.

Recordando el modelo que comenté en el capítulo 9 (con las "largas líneas de caracteres"), vamos a volver a analizar la generación de una columna de posiciones de impresión en la pantalla.

Al realizar una columna negra o un grupo de ellas en la pantalla, estás introduciendo en una serie de posiciones del fichero de imagen los valores FFh (255d). Si comenzamos

por la parte superior izquierda de la pantalla, la primera dirección es la 4000h (16384d). Después tenemos que añadir a esta dirección el valor 100h (256d) para acceder a las siete posiciones siguientes, las cuales son el comienzo de cada línea de imagen, de las ocho que forman la larga línea de caracteres. Después de esto, tenemos que volver a la posición 32 de la primera línea larga para acceder a la siguiente posición vertical, situada en la pantalla debajo de la última ya dibujada (vuelve a observar la figura 9.1 del capítulo 9).

Tras esto, realizamos la misma operación anterior de añadir el valor 100h, ocho veces más, y volveremos a comenzar el proceso en una posición 40h mayor que el comienzo de la actual línea, para acceder al siguiente grupo de líneas que forman el carácter situado debajo del último dibujado; el proceso de añadir 100h a esta dirección se repetirá ocho veces.

Hecho esto, comenzaremos a trabajar con el siguiente grupo de líneas del fichero de imagen que forman la siguiente línea larga imaginaria, comenzando por la dirección de base, más un desplazamiento de 800h (2048d). Y después continuaremos con la tercera y última de estas largas líneas imaginarias.

En realidad, el programa en BASIC que realiza esto no tiene tan mal aspecto como pudieras imaginarte al leer los párrafos anteriores.

### Introducción de una columna vertical en la pantalla

```
10 LET x=16384
20 FOR z=0 TO 2
30 FOR i=0 TO 7
40 FOR j=0 TO 7
50 POKE x+256*j+32*i,255
60 NEXT j: NEXT i
70 LET x=x+2048: NEXT z
```

Puedes ver cómo hay tres bucles anidados, de 8 × 8 y tres pasadas, respectivamente. Esto mismo se realiza en la versión en código máquina, con la diferencia de que, en este segundo caso, el control de los contadores de cada bucle se realiza con operaciones lógicas de suma (ADD), por lo que no resultan tan sencillas de apreciar a simple vista.

### Impresión de una columna vertical

```
E000           10      ORG   #E000
E000 211040    20      LD    HL,#4010
E003 06C0      30      LD    B,#C0
E005 3EFF      40      LD    A,#FF
E007 77        50      LD    (HL),A
E008 24        60      INC   H
E009 7C        70      LD    A,H
E00A E607      80      AND   #07      ;Chequea si parte
               90 ;                     inferior de linea
```

```
                               larga
E00C 200A       JR       NZ,##E018
E00E 7D         LD        A,L
E00F C620       ADD      A,#20
E011 6F         LD        L,A      ;Pasa a sig. posicion
E012 3F         CCF
E013 9F         SBC      A,A
E014 E6F8       AND      #F8        ;Correccion para cada
                              tercio de pantalla
E016 84         ADD      A,H
E017 67         LD        H,A
E018 10EB       DJNZ     #E005
E01A C9         RET
```

## Búsqueda de los UDG's

Antes de entrar en lo que a cierta gente le gusta llamar "análisis en profundidad de la situación de la imagen en pantalla", el método de direccionamiento que he utilizado anteriormente sirve para ayudar en una situación en la que el Spectrum falla.

Puedes leer, en el manual de *Sinclair,* que la opción SCREEN$ no reconoce los caracteres definibles por el usuario, aquí llamados UDG's. Esto nos supone un problema, ya que ésos son los caracteres que utilizamos normalmente en nuestros gráficos. Pero ahora sabemos cómo superar este problema; si realizamos adecuadamente los *sprites,* podremos diferenciarlos con sólo mirar su línea superior de bits.

Recordando la baraja de cartas sobre la que tratamos en otro capítulo anterior, he realizado algunos diseños de lo que podrían ser los cuatro palos de la baraja, los cuales pueden ser introducidos en los UDG's. Su aspecto final es el que puedes ver en la figura 12.1.

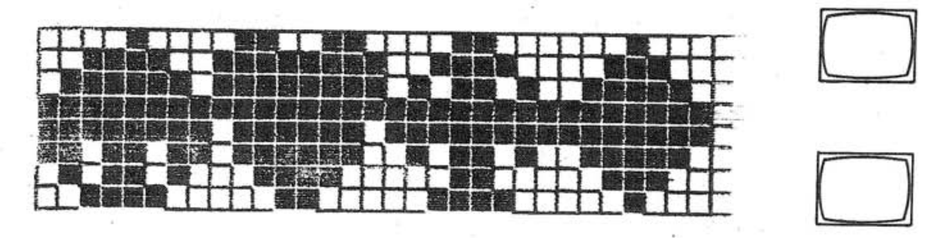


*Fig. 12.1. Diseño de los palos de una baraja*

Si observas estos caracteres gráficos, verás que la línea superior de las ocho que los forman es diferente para cada gráfico, lo que supone que, cuando están en el fichero de imagen, serán identificables según el contenido del byte superior de la posición de impresión que ocupen. A continuación puedes ver el valor de estos bytes para cada gráfico:

| | | |
|---|---|---|
| Pica: | 08h | 8d |
| Corazón: | 66h | 102d |
| Trébol: | 18h | 24d |
| Diamante: | 10h | 16d |

Puedes buscar los gráficos de los palos de la baraja con el programa siguiente; este programa dibujará un cuadrado de color en la primera posición que se encuentre con uno de estos gráficos. Da la casualidad de que ninguno de los caracteres del Sinclair utiliza estos valores en su primer byte, ya que la mayoría de ellos tienen este byte a cero.

**Búsqueda de la posición que ocupa un palo de la baraja**

```
   5 PRINT AT 8,5;"♠ ♥ ♣ ♦"
  10 FOR i=0 TO 2: FOR j=0 TO 25
  20 IF PEEK (16384+j+i*2048)=16
 THEN  GO TO 40
  30 NEXT j: NEXT i
  40 PRINT AT i*8+INT (j/32),j-3
2*INT (j/32); OVER 1; PAPER 4;"
"
```

Sin embargo, tú estás más interesado en querer identificar el palo situado en una posición determinada. En este caso deberías de utilizar una rutina como la siguiente, la cual añadirá el color adecuado al carácter gráfico situado en la posición determinada.

**Análisis de la línea superior**

```
   1 REM x=Columna y=Linea
1010 LET zona=INT (y/8)
1020 LET y1=y-8*zona
1030 LET p=16384+zona*2048+y1*32
+x
1040 PRINT  INK 2 AND (PEEK p=16
 OR PEEK p=102)
```

La variable "zona" identifica en qué tercio de la pantalla estamos.

## Deslizamiento del contenido de la pantalla ("scrolling")

Hay otra operación que puedes intentar realizar con el fichero de imagen y que no resulta muy complicada; esta operación es el desplazamiento del fichero de imagen a la izquierda o a la derecha. Me estoy refiriendo a un desplazamiento a nivel de puntos de

imagen, en vez de posiciones completas de impresión (conjuntos de 8 × 8 puntos de imagen), aunque esta aclaración no supone que sea más complicado de realizar de esta manera.

La razón por la cual no resulta difícil realizar el deslizamiento de puntos de imagen es que, como no estás cambiando las posiciones de la línea, no importa el orden en que los tomes; tú puedes comenzar por la parte superior y acabar en la inferior.

Para realizar el deslizamiento has de tomar los bytes del fichero de imagen de uno en uno y de forma ordenada (direccionando el byte adecuado con el registro HL), luego rotar el registro HL (a la izquierda o a la derecha) y tomar el siguiente byte así direccionado. Lo único que has de recordar es que debes comenzar por el *final,* si estás deslizando el fichero hacia la izquierda, o por el *comienzo,* si estás deslizándolo a la derecha. Esto se hace para permitir que los bits que se desbordan fuera de cada carácter, por sus bordes, sean recogidos y colocados en la posición adecuada del próximo carácter.

A continuación, encontrarás la forma más sencilla de realizar el desplazamiento. Para ello has de crearte dos bucles (es decir, hacer el desplazamiento línea a línea), de forma que tienes la posibilidad de no realizar el último movimiento de la línea. Si no lo realizas, la pantalla se deslizará en dirección contraria a como lo hacía, volviendo a posiciones muy particulares. Puedes intentarlo tú mismo, sustituyendo el valor "00" por el "AF", en la posición F007h.

**Deslizamiento de la pantalla hacia la izquierda**

```
F000              10       ORG    #F000
F000 21FF57       20       LD      HL,#57FF
F003 0EC0         30       LD      C,#C0
F005 0620         40       LD      B,#20
F007 AF           50       XOR    A
F008 CB16         60       RL      (HL)
F00A 2B           70       DEC    HL
F00B 10FB         80       DJNZ   #F008
F00D 0D           90       DEC    C
F00E 20F5        100       JR      NZ,#F005
F010 C9          110       RET
```

Para realizar el desplazamiento en la otra dirección, has de cambiar la primera línea, de forma que quede formada por "21 00 40"; cambiar la línea situada en F008h a "CB 1E" y la instrucción que le sigue cambiarla de "2B" a "23".

El desplazamiento del contenido de la pantalla no resulta útil si no tienes preparado nada con lo que realizar el desplazamiento. Esto quiere decir que has de tener otro fichero de imagen preparado en otra posición de memoria, para que el desplazamiento de la imagen produzca unos efectos apreciables en la pantalla.

Para que te sirva de ejemplo, a continuación tienes una rutina para deslizar por la pantalla dos escenas de una forma continua, produciendo el efecto que puedes asociar al movimiento del tambor de una lavadora. Esta rutina puede serte útil para producir un fondo de imagen móvil.

### Deslizamiento cíclico, derecha a izquierda

```
F000             10        ORG    #F000
F000 11FFE7      20        LD      DE,#E7FF ;<=
F003 21FF57      30        LD      HL,#57FF ;<=
F006 F5          40        PUSH   AF
F007 0EC0        50        LD      C,#C0
F009 F1          60        POP    AF
F00A E5          70        PUSH   HL
F00B 0620        80        LD      B,#20
F00D AF          90        XOR    A
F00E 7E         100        LD      A,(HL)
F00F 17         110        RLA              ;<=
F010 77         120        LD      (HL),A
F011 2B         130        DEC    HL        ;<=
F012 10FA       140        DJNZ   #F00E
F014 0620       150        LD      B,#20
F016 1A         160        LD      A,(DE)
F017 17         170        RLA              ;<=
F018 12         180        LD      (DE),A
F019 1B         190        DEC    DE        ;<=
F01A 10FA       200        DJNZ   #F016
F01C E3         210        EX      (SP),HL
F01D 3E00       220        LD      A,00
F01F 17         230        RLA              ;<=
F020 B6         240        OR      (HL)
F021 77         250        LD      (HL),A
F022 E3         260        EX      (SP),HL
F023 F1         270        POP    AF
F024 F5         280        PUSH   AF
F025 0D         290        DEC    C
F026 20E1       300        JR      NZ,#F009
F028 F1         310        POP    AF
F029 C9         320        RET
```

### Transferencia del segundo fichero de imagen a la posición D000h

```
F030            400        ORG    #F030
F030 210040     410        LD      HL,#4000
F033 1100D0     420        LD      DE,#D000
F036 010018     430        LD      BC,#1800
F039 EDB0       440        LDIR
F03B C9         450        RET
```

### Deslizamiento completo de la pantalla

```
F040             10        ORG    #F040
F040 010000      20        LD      BC,#0000
F043 C5          30        PUSH   BC
F044 CD00F0      40        CALL   #F000
F047 C1          50        POP    BC
F048 10F9        60        DJNZ   #F043
F04A C9          70        RET
```

El programa principal está formado, en realidad, por la unión de dos versiones distintas del programa anterior. El fichero de imagen que se observa en la pantalla está colo-

cado en la dirección 4000h y el segundo está situado en la posición D000h. El registro HL contiene la dirección del primer fichero y el registro DE contiene la dirección del segundo. Recuerda que, para hacer el deslizamiento en la dirección indicada, hemos de comenzar por el final del fichero, por lo que las direcciones iniciales serán 57FFh y E7FFh.

Hay una operación interesante en las posiciones F01Ch a F024h del programa. Al final de la línea nos encontramos con un bit de acarreo que tenemos que tratar. Este bit ha resultado desbordado del último byte de la línea y ha de ser añadido al primer byte de la misma línea, si quieres que se realice un deslizamiento continuo. Para hacer esto, tenemos que introducir la dirección del primer byte en el *stack,* al comienzo de la rutina (en la posición F00Ah es donde esto se realiza), ya que cuando llegamos a la posición F01Ch introducimos una nueva dirección en el registro HL; con esta dirección será con la que trabajará el resto de la rutina. No se introduce en el *stack* el nuevo valor del registro HL, por lo que tenemos que operar con esta nueva dirección y la primera, la cual aún permanece en la parte alta del *stack*. En la direccion almacenada en el *stack* será donde se introduzca el bit de acarreo. Después de esto, cambiamos de nuevo la dirección contenida en el registro HL y nos deshacemos de la anterior (la dirección del primer byte), introduciéndola en AF (mediante el uso de un POP), continuando después con el trabajo.

Las flechas que aparecen en la rutina señalan los bytes que han de ser cambiados, si es que deseas cambiar la dirección en que se realiza el desplazamiento *(scroll)*. A continuación puedes encontrar estos cambios tabulados según la dirección de desplazamiento deseada:

| | *Dcha.-izq.* | *Izq.-dcha.* |
|---|---|---|
| F000-02 | 11 FF E7 | 11 00 D0 |
| F003-05 | 21 FF 57 | 21 00 40 |
| F00F | 17 RLA | 1F RRA |
| F011 | 2B DEC HL | 23 INC HL |
| F017 | 17 RLA | 1F RRA |
| F019 | 1B DEC DE | 13 INC DE |
| F01F | 17 RLA | 1F RRA |

Las otras dos rutinas pequeñas se utilizan en conjunción con el programa principal. La primera transfiere lo que haya en la pantalla al segundo fichero de imagen que hemos creado a partir de la posición D000h. La otra actúa como programa de demostración, que realizará un desplazamiento desde una imagen a la otra (desde la que está en la pantalla a la anteriormente almacenada); esta operación tarda en realizarse 256 ciclos (desde que el registro B vale 0 hasta que vuelve a valer 0).

## Reorganización del fichero

El desplazamiento lateral de la pantalla, tal y como habrás observado, es una operación que no resulta demasiado complicada. Sin embargo, si queremos que el desplazamiento se realice hacia arriba o hacia abajo, volvemos a tener problemas con la ejecución de los saltos de línea a línea. Después de trabajar algún tiempo sobre la solución de este problema, he llegado a la conclusión de que el método más razonable para resolver-

lo consiste en copiar todo el fichero de imagen en otra posición de memoria y reorganizarlo de una manera que nos resulte más fácil manejarlo. Esta nueva organización consiste en colocar la línea de imagen dos veces después de la 1, la 3 detrás de la 2, y así sucesivamente.

Con esta organización sólo tendremos que utilizar dos pequeñas rutinas para cubrir todas las posibilidades; una, para reorganizar los datos en la forma que deseemos y, la otra, para reconstruir el fichero al formato necesario para su representación en pantalla, para posteriormente sacarlo a pantalla.

Las rutinas son bastante parecidas a las que utilizamos al principio de este capítulo para realizar las columnas verticales, con la diferencia de que ahora se trabaja con líneas completas de bytes, en vez de con un byte cada vez. La primera de las rutinas es:

**Reorganización del fichero de imagen en la zona de memoria D000-D800h**

```
F000                ORG   #F000
F000 210040         LD     HL,#4000
F003 1100D0         LD     DE,#D000
F006 0EC0           LD     C,#C0
F008 E5             PUSH  HL
F009 0620           LD     B,#20
F00B 7E             LD     A,(HL)
F00C 12             LD     (DE),A
F00D 23             INC   HL
F00E 13             INC   DE
F00F 10FA           DJNZ  #F00B
F011 ED00
F013 E1             POP   HL
F014 24             INC   H
F015 7C             LD     A,H
F016 E607           AND   #07
F018 2008           JR     NZ,#F022
F01A 7D             LD     A,L
F01B C620           ADD   A,#20
F01D 6F             LD     L,A
F01E 3F             CCF
F01F 9F             SBC   A,A
F020 E6F8           AND   #F8
F022 84             ADD   A,H
F023 67             LD     H,A
F024 0D             DEC   C
F025 20E1           JR     NZ,#F008
F027 C9             RET
```

La realización del programa para reconstruir el fichero de imagen es incluso más fácil: haces una copia del primer programa en una nueva dirección (yo he utilizado la F030h). Si dispones de un buen ensamblador o de un programa de edición, probablemente puedas hacer todo esto de una forma automática. Hecho esto, modificas dos instrucciones: "LD A (HL)", de la posición F03Bh, la cambias por la "LD A (DE)" y la que está en la posición F03Ch, "LD (DE) A", por la "LD (HL) A".

**Reconstrucción del fichero de imagen basándonos en la copia que hay en la zona comprendida entre D000-D800h**

```
F030              ORG   #F030
F030 210040       LD     HL,#4000
F033 1100D0       LD     DE,#D000
F036 0EC0         LD     C,#C0
F038 EDCB00
F03B E5           PUSH  HL
F03C 0620         LD     B,#20
F03E 1A           LD     A,(DE)
F03F 77           LD     (HL),A
F040 23           INC   HL
F041 13           INC   DE
F042 10F7         DJNZ  #F03B
F044 00
F045 E1           POP   HL
F046 24           INC   H
F047 7C           LD     A,H
F048 E607         AND   #07
F04A 2006         JR     NZ,#F052
F04C CB00
F04E 7D           LD     A,L
F04F C620         ADD   A,#20
F051 6F           LD     L,A
F052 3F           CCF
F053 9F           SBC   A,A
F054 E6F8         AND   #F8
F056 84           ADD   A,H
F057 67           LD     H,A
F058 0D           DEC   C
F059 20DD         JR     NZ,#F038
F05B C9           RET
```

Utilizando estas dos rutinas, podemos realizar una gran cantidad de manipulaciones de la pantalla. El desplazamiento de la imagen arriba y abajo, punto a punto, se realiza de una forma sencilla con la instrucción LDIR. Podrás también realizar de una forma sencilla desplazamientos de secciones de la pantalla (no tienen por qué ser secciones de un tercio de la pantalla, correspondientes al espacio ocupado por cada línea imaginaria del fichero de imagen), e incluso puedes trabajar con ventanas de la imagen (zonas de la imagen, de cualquier tamaño y situadas en cualquier parte de la pantalla).

Resultan igualmente sencillos los desplazamientos completos de las imágenes en pantalla; podrías reemplazar la rutina del desplazamiento cíclico que vimos anteriormente, pero en este caso tendrías que utilizar dos ficheros de imagen al mismo tiempo y esto parece demasiado complicado para lo que intentamos realizar en este momento.

A continuación hay un programa para realizar un desplazamiento en diagonal de la imagen, con una rutina para realizar este desplazamiento de forma continuada.

**Desplazamiento en diagonal de la pantalla**

```
F100              ORG   #F100
F100 2120D0       LD     HL,#D020
F103 1100D0       LD     DE,#D000
F106 01E017       LD     BC,#17E0
```

```
F109 CB1E         50        RR      (HL)
F10B EDA0         60        LDI
F10D EA09F1       70        JP      PE,#F109
F110 C330F0       80        JP      #F030
```

**Rutina para la repetición del desplazamiento**

```
F200              10        ORG     #F200
F200 06A0         20        LD       B,#A0
F202 C5           30        PUSH    BC
F203 CD00F1       40        CALL    #F100
F206 C1           50        POP     BC
F207 10F9         60        DJNZ    #F202
F209 C9           70        RET
```

Espero que no hagas como hice yo, al olvidar incluir las instrucciones "PUSH BC" y "POP BC", para así guardar el contador en el segundo programa; con esto presenciarás cómo el listado completo desaparece por la esquina derecha.

## Reducción de la imagen

Hasta ahora, sólo hemos visto pequeñas muestras de la libertad de diseños que nos ofrece nuestro fichero de imagen con su nueva organización. No hay grandes problemas para realizar la reducción de la imagen a un tamaño que es la cuarta parte del original, tal y como se muestra en la figura 12.2.

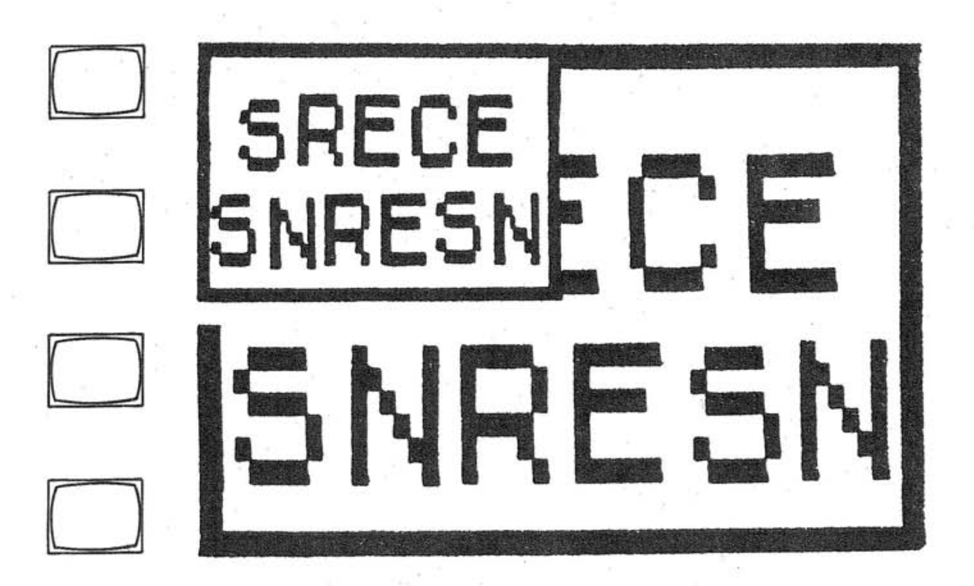
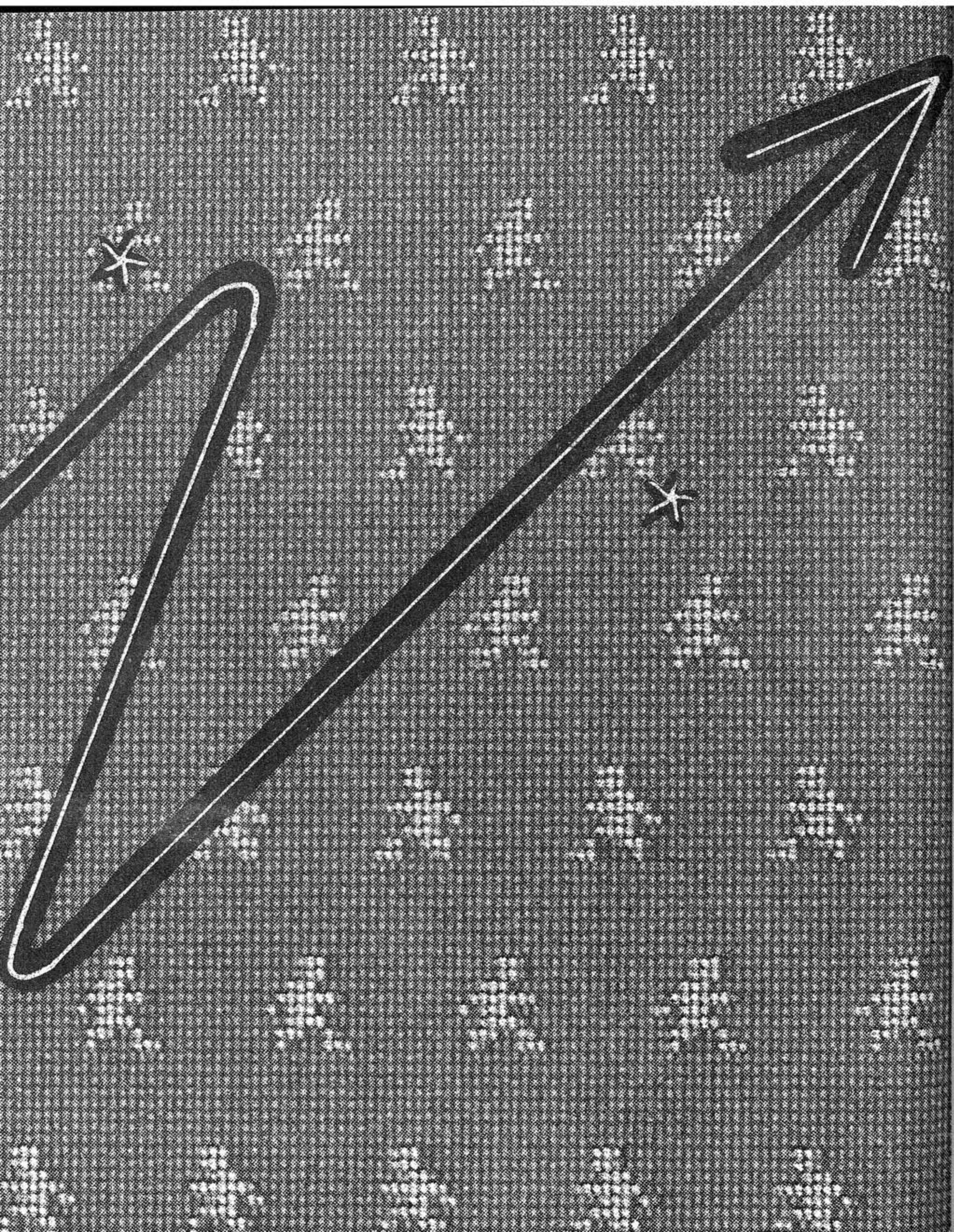



*Fig. 12.2*

La rutina que aparece a continuación trabaja de una forma poco depurada (con una resolución muy baja), ya que comprime todo lo que hay en la pantalla en una zona situada en la esquina derecha de dicha pantalla, utilizando instrucciones de rotación decimal a la izquierda y saltando todas las líneas alternativas (de cada dos líneas sólo dibujará una). Sólo trabajará con unos caracteres gruesos y claros, tales como los que he usado en la figura 12.2. Sin embargo, no resulta complicado extraer los bits del fichero original, de una forma alternativa, para dar una mejor resolución que trabajando con líneas alternativas, tal y como hicimos para generar los caracteres de cuatro bits. Sin embargo, pienso que la lectura no resultaría tan clara utilizando caracteres comprimidos a una matriz de 4 × 4.

### Realización de una reducción de la pantalla a un cuarto de su tamaño original

```
F060              ORG    #F060
F060 1100D0       LD      DE,#D000
F063 2100D0       LD      HL,#D000
F066 0E60         LD      C,#60
F068 0610         LD      B,#10
F06A 1A           LD      A,(DE)
F06B ED6F         RLD
F06D 13           INC    DE
F06E 1A           LD      A,(DE)
F06F ED6F         RLD
F071 13           INC    DE
F072 23           INC    HL
F073 10F5         DJNZ   #F06A
F075 C5           PUSH   BC
F076 011400       LD      BC,0020
F079 EB           EX      DE,HL
F07A 09           ADD    HL,BC
F07B EB           EX      DE,HL
F07C 0E10         LD      C,#10
F07E 09           ADD    HL,BC
F07F C1           POP    BC
F080 0D           DEC    C
F081 20E5         JR      NZ,#F068
F083 C330F0       JP      #F030
```



# 13
# Entradas y salidas

Como ya he comentado en ocasiones anteriores, con frecuencia me olvido de que el Spectrum no es un sistema completo y autónomo *(self-contained)*. En realidad, si no fuera por una serie de entradas y salidas de las que se sirve, no tendría un uso práctico.

Los sistemas normales de entrada son el teclado y el magnetófono o el *microdrive*. Los sistemas de salida son el televisor, el magnetófono o el *microdrive*, con el generador de tonos (formando conjunto con el altavoz) y la impresora. Se le pueden incorporar una gran cantidad de equipos periféricos de entrada/salida (I/O), tales como *joysticks*. Pero, para poder utilizarlos, todos ellos necesitan adaptadores especiales (tanto a nivel de *hardware* como de *software,* los llamados *interfaces*), y en este libro voy a analizar los métodos utilizados para el uso de los diferentes equipos que componen la configuración típica del Spectrum de una manera en la que no pensó la Sinclair Research.

Puedo continuar diciendo que no se hace nada en el televisor o en el magnetófono con la información que le mandamos desde el ordenador. Tampoco se mejora mucho utilizando la salida a la impresora, pero existe el peligro de colgar el ordenador en un bucle, por lo que no trabajaré con estos aparatos en este capítulo.

Los temas que creo que merece la pena ser tratados aquí son los referentes a la salida que afecta a la parte más externa de la pantalla, la cual se controla normalmente con el comando "BORDER X" y el generador de tonos.

Incluso en estos temas, las posibilidades son limitadas. Puedes, por ejemplo, crear un efecto como si explotase la imagen. O también puedes modificar los tonos generados por BEEP para crear una escala de silbidos. Para hacer que alguna de estas posibilidades sea más eficaz, podemos incluso cambiar el modo de interrupción.

Voy a explicar este tema de las interrupciones en primer lugar.

## Modos de interrupción

En un trabajo normal, el *hardware* (la parte electrónica) del Spectrum interrumpe el trabajo que está realizando el microprocesador Z80 cada 20 microsegundos. Esto significa que el microprocesador detiene la ejecución de lo que esté haciendo y salta a una rutina de tratamiento de interrupciones, la cual ha de ser ejecutada completamente antes de volver a la tarea que estaba realizando en un principio.

Esta rutina de interrupción está situada de una forma fija en la posición 0038h de la ROM y realiza una rápida inicialización del contador interno del Spectrum denominado "FRAMES"; a continuación explora el teclado para ver qué teclas han sido presionadas.

El microprocesador Z80 ha sido diseñado para permitir que se produzcan estas interrupciones, para así poder realizar este tipo de operaciones regularmente. Este microprocesador tiene, en realidad, tres tipos (modos) de interrupciones, conocidas como IM0, IM1 e IM2. Posee también una interrupción no enmascarable, es decir, que siempre que se produzca será atendida por el microprocesador; es conocida como NMI.

La interrupción IM0 no puede ser utilizada por el Spectrum. La IM1 es la utilizada normalmente para explorar el teclado, etc.: provoca un salto incondicional al comienzo de la rutina situada en la posición 0038h; ésta es la de la instrucción "RST 38". La interrupción NMI ha sido bloqueada en la ROM del Sinclair. Por tanto, sólo se puede utilizar la IM2.

Vamos a ver lo que hace la interrupción IM2. Cuando el microprocesador ha sido programado para tratar la interrupción IM2 y ésta se produce, el Z80 pasa directamente a una dirección determinada. Esta dirección está compuesta de un byte menos significativo, tomado de un dispositivo de los que forman el Sinclair (la ULA[1] del Sinclair) y un byte más significativo que se toma del contenido del registro I, que es un registro que tiene el microprocesador para realizar esta operación.

Uniendo estos dos bytes formamos la dirección a la que saltará el Sinclair, en donde el ordenador espera encontrar otra dirección que habrá sido colocada en esa posición por el programador. El microprocesador procede, después de esto, a ejecutar cualquier rutina que se encuentre en esta última dirección.

Esto te podrá parecer muy complicado, pero el propósito de todo esto es permitir al microprocesador emprender la ejecución de rutinas particulares indicadas por un determinado periférico.

Ahora puede que te preguntes cómo podemos aprovechar todo esto. Cuando se produce una interrupción, la ULA no está programada para proporcionar un byte determinado, por lo que el byte generado por defecto debería ser "FF".

Esto significa que la dirección que se generará al aparecer la interrupción IM2 será del tipo "xxFF", donde "xx" será lo que el microprocesador encuentre en el registro I, y éste sí que es un valor que podemos controlar nosotros.

Por razones complicadas del *hardware,* la dirección completa deberá estar en la zona comprendida entre 4000h y 7FFFh, la cual supone toda la memoria RAM disponible por los usuarios de los Spectrum de 16K de memoria. Sin embargo, los poseedores de Spec-

[1] La ULA es la Unidad Aritmético-lógica del ordenador; es el dispositivo en el que se efectúan todas las operaciones que realiza el ordenador, tanto las aritméticas como las lógicas.

trum de 16K no han de desesperarse, ya que posteriormente intentaremos solucionar este problema.

Los que posean Spectrum de 48K de memoria tienen libertad para elegir cualquier dirección con el formato "xxFF" superior a la 8000h y almacenar en ella la dirección de la rutina de interrupción que ellos han escrito. Por ejemplo, si tu rutina de tratamiento de la interrupción está en la posición F001h, entonces podrás colocar esta dirección en la posición EFFFh en la forma mostrada a continuación:

| | |
|---|---|
| EFFF | 01 |
| F000 | F0 |

Todo lo que tienes que hacer es fijar el modo de interrupción a IM2, cargar, en nuestro caso, el registro I con el byte "EF" y preparar una rutina adecuada para tratar esta interrupción a partir de la posición F001h. El Spectrum pasará a ejecutar esta rutina cada vez que se produzca la interrupción, deteniendo cualquier otra cosa que estuviera ejecutando.

Si deseas que esta rutina controle el reloj y chequee el estado del teclado, entonces tendrás que saltar a la posición 0038h cuando finalice la rutina de interrupción que estás ejecutando. Otra forma de realizar todo esto consiste en colocar una situación en que se permitan las interrupciones (colocando el byte "FB" en el registro I), seguida de una instrucción RET o una RETI.

Un punto importante a resaltar es que *debes salvar todos los registros* que van a ser utilizados en la rutina de tratamiento de las interrupciones, incluyendo entre ellos el AF y los registros alternativos si son utilizados. Todos estos registros han de ser almacenados en el *stack* al entrar en la rutina de tratamiento de las interrupciones y sacados del *stack* antes de finalizar la rutina.

Los propietarios de modelos con 16K de memoria pueden sentirse un poco desanimados al haber leído los párrafos anteriores, pero hay una forma de superar los problemas de falta de memoria que les permitirá continuar con la lectura de este tema. La forma de superarlo consiste en utilizar una dirección de la ROM. Como ya sabes, la memoria ROM no puede modificarse, pero puedes buscar dos bytes adyacentes que estén en las posiciones "xxFF" y la "xxFF+1" y que contengan, entre ambos, una dirección válida de la memoria RAM.

Esta búsqueda a través de la memoria ROM proporcionará los siguientes resultados cuando buscamos direcciones de la RAM mayores que A000h situadas en posiciones de la ROM del tipo "xxFFh":

**Direcciones de la ROM del tipo "xxFF" en el modelo de 48K**

| *Dirección decimal* | *Direc. hexadecimal* | *Direc. formada* |
|---|---|---|
| 511 | 01FF | CE52 |
| 2559 | 09FF | FE69 |
| 3071 | 0BFF | E608 |
| 3327 | 0CFF | CFBF |

| *Dirección decimal* | *Direc. hexadecimal* | *Direc. formada* |
|---|---|---|
| 3583 | 0DFF | CD17 |
| 4351 | 10FF | CB10 |
| 4863 | 12FF | CD01 |
| 5119 | 13FF | C255 |
| 5631 | 15FF | C9D9 |
| 5887 | 16FF | C970 |
| 7423 | 1CFF | C31B |
| 8447 | 20FF | CD21 |
| 9215 | 23FF | C181 |
| 10751 | 29FF | E32A |
| 11775 | 2DFF | D9E5 |
| 12543 | 30FF | EB30 |
| 12799 | 31FF | E128 |
| 13823 | 35FF | DF24 |
| 14335 | 37FF | A10F |
| 14591 | 38FF | FFFF |
| 14847 | 39FF | FFFF |
| 15103 | 3AFF | FFFF |
| 15359 | 3BFF | FFFF |
| 23551 | 5BFF | FF00 |

La siguiente lista es la que resulta de la inspección del Spectrum de 16K:

**Direcciones de la ROM del tipo "xxFF" en el modelo de 16K**

| | | |
|---|---|---|
| 1791 | 71DD | 06FF |
| 4095 | 0FFF | 6D18 |
| 5375 | 14FF | 6469 |
| 7935 | 1EFF | 67CD |
| 10495 | 28FF | 7E5C |
| 24063 | 5DFF | 6964 |
| 24831 | 60FF | 79A2 |

Hay una gran cantidad de posibilidades. La dirección útil más alta, en el caso del Sinclair de 48K, es la "FE69h", que está almacenada en las posiciones de memoria 09FFh y 0A00h. En el caso del Sinclair de 16K, la dirección útil más alta es la 7E5Ch, que está almacenada en las posiciones "28FFh" y "2900h".

En el párrafo anterior me refería a la dirección útil más alta, ya que, aunque en el caso del Sinclair de 48K los propietarios del mismo disponen, según las anteriores tablas, de la dirección final (la FFFFh) como una posibilidad más para este propósito, la dirección FFFFh direcciona a la última posición de la RAM, por lo que la siguiente dirección ya no caerá en la parte de la memoria utilizable por el usuario (la RAM), sino que la dirección siguiente es la 0000h que forma parte de la ROM. Con el contenido de este par de direcciones formamos una dirección utilizable para colocar en ella la rutina de tratamiento de la interrupción; esta dirección es del formato "F3xxh".

Como puedes ver, puedes cambiar la parte baja de esta dirección al ser éste el byte que está contenido en la posición "FFFFh", que aún es parte de la RAM.

| *Dirección* | *Contenido* |
|---|---|
| FFFF | XX |
| 0000 | F3 |

Hay otros dos puntos en los que hay que tomar ciertas precauciones debido, sobre todo, a que estamos utilizando el Spectrum para cosas que su creador, Sir Clive, no había previsto. En primer lugar, no existe ningún sitio donde se indique que el byte menos significativo de la dirección del *buffer* de entrada ha de ser "FF". Normalmente lo es, pero, si tú quieres, este valor puede ser cambiado. Podría ser un valor cualquiera. Todos estos problemas no se producen al trabajar con el *microdrive*.

Continuando con este problema, podrías crearte un conjunto completo de bytes (257 en total) en el que cualquier pareja de ellos dieran como resultado la misma dirección. Por ejemplo, podrías escoger el byte "FE". Formando parejas con este byte, tendrías la dirección "FEFE", en la que podrías colocar tu rutina de tratamiento de interrupciones.

Deberías cargar todos los bytes situados en la zona de memoria creada, por ejemplo, desde la posición FD00h a la FE00h, con el valor FE. Después cogerías el byte FD y lo introducirías en el registro I, con lo que no importaría el byte que cogiéramos para completar la dirección de base, ya que siempre resultaría una dirección FEFE al juntar los bytes contenidos en la dirección "FDxx" y la siguiente, al estar ambas posiciones en la zona de memoria que anteriormente rellenamos con el byte FE.

En segundo lugar has de tener mucho cuidado cuando utilices direcciones de la ROM estando conectado el Interfaz 1[1]. El Spectrum puede suponer que la dirección base se refiere a una posición de la ROM del Interfaz 1, produciéndose entonces unos resultados imprevisibles. No conozco otra manera de evitar que surja este problema que no sea el desconectar el Interfaz.

En este momento hemos llegado al punto en el que deseamos saber cómo fijar el modo de interrupciones para colocarlo en el registro I. La rutina siguiente es del tipo de las que utilizan una dirección de la zona de memoria ROM.

**Rutina para fijar el modo de Interrupción 2**

```
F010            10      ORG   #F010
F010 3E3F       20      LD     A,#3F
F012 ED47       30      LD     I,A
F014 ED56       40      IM   1
F016 C9         50      RET
```

También necesitaremos una rutina parecida, en código máquina, para volver al estado en que estaba el ordenador, una vez que se haya terminado de ejecutar nuestro programa de tratamiento de la interrupción esperada.

[1] Interfaz: conjunto de necesidades *hardware* y *software* para comunicarse con los dispositivos periféricos de un ordenador.

**Restablecimiento del modo de Interrupción 1**

```
F000              10         ORG    #F000
F000 3E09         20         LD      A,#09
F002 ED47         30         LD      I,A
F004 ED5E         40         IM   2
F006 C9           50         RET
```

Los programas que colocamos en la dirección de interrupción FE69h pueden ser de cualquier tipo. Sin embargo, has de recordar que el microprocesador Z80 no puede volver a ejecutar la rutina principal hasta que no haya finalizado la ejecución de la rutina para el tratamiento de las interrupciones, por lo que la rutina principal se ejecutará más lentamente.

El problema principal que aparece al trabajar con el conmutador de interrupciones y que limita su utilización, es que nosotros lo imaginamos de una forma equivocada. Lo que deseas es controlar el tiempo de la interrupción, tanto como la rutina de interrupción: aquí aparece, por tanto, un límite para el número de cosas que quieres controlar de una forma regular, 50 veces por segundo. Nos resultaría mucho más útil el tener la interrupción bajo nuestro control, para así poder aprovechar completamente sus posibilidades. Pero esto no es posible, por lo que nos debemos conformar con lo que tenemos.

## La pantalla de televisión

A continuación tienes un programa que transforma el borde de la pantalla en una especie de helado napolitano.

**Borde de la pantalla con apariencia de helado napolitano**

```
FE69              10         ORG    #FE69
FE69 F5           20         PUSH   AF
FE6A C5           30         PUSH   BC
FE6B 0608         40         LD      B,#08
FE6D 0E00         50         LD      C,#00
FE6F 78           60         LD      A,B
FE70 3D           70         DEC    A
FE71 D3FE         80         OUT    (#FE),A
FE73 00           90         NOP
FE74 00          100         NOP
FE75 00          110         NOP
FE76 0D          120         DEC    C
FE77 20F9        130         JR      NZ,#FE72
FE79 10F2        140         DJNZ   #FE6D
FE7B C1          150         POP    BC
FE7C F1          160         POP    AF
FE7D C33800      170         JP      #0038
```

Las operaciones NOP que aparecen en el programa se utilizan para crear unos retardos, con lo que los colores son creados de una forma muy suave.

Otra variación divertida es la siguiente: Hacer que parezca que la pantalla explota. Esto se realiza haciendo que los colores cambien cada vez que aparece una nueva imagen en la pantalla.

**Destellos del borde de la pantalla**

```
FE69                ORG     #FE69
FE69 F5             PUSH    AF
FE6A 3A815C         LD      A,(#5C81)

FE6D D3FE           OUT     (#FE),A
FE6F 3D             DEC     A
FE70 2002           JR      NZ,#FE74
FE72 3E08           LD      A,#08
FE74 32815C         LD      (#5C81),A

FE77 F1             POP     AF
FE78 C33800         JP      #0038
```

Probablemente la aplicación más útil del cambio de modo de interrupción es la de permitirte que te crees un fondo móvil, de forma totalmente independiente a otros programas que estén ejecutándose en el transcurso de un juego (tal y como sucede con el desplazamiento continuo de la imagen, descrito en el capítulo 12).

## El generador de tonos del Spectrum

Hay otra aplicación en la que merece la pena utilizar algunas técnicas de entrada/salidad, y ésta es la utilización del generador de tonos (BEEP).

En una aplicación normal, el generador de tonos producirá un tono único y de una determinada duración. Utilizando la programación en BASIC y los bucles FOR...NEXT, puedes obtener una sucesión de varias notas, pero no puedes obtener un deslizamiento suave entre los tonos, ya que el conmutador utilizado en BASIC para los lazos rompe la continuidad; sin embargo, algunos programas sencillos en código máquina hacen que esto sea posible.

El Spectrum crea los tonos de salida de una forma muy sencilla. El altavoz interno está conectado a uno de los puertos de salida del procesador Z80 (véase *Manual del Spectrum*). Cuando el bit del altavoz (D4) está a uno, se activa el circuito y se produce un *click* en el altavoz. Si hacemos que el bit que actúa como conmutador (D4) cambie de estado algunos cientos de veces por segundo, el oído lo interpretará como un tono determinado.

Como fácilmente comprenderás, con este sistema de generación de tonos no hay manera de modificar las formas de onda, y por tanto de cambiar el volumen y las características de ésta, sin añadir circuitos adicionales a los que posee el Spectrum. Sin embargo, hay una característica de la señal que podemos manejar: el tono; podemos, y hacemos, variar la frecuencia con que se producen los *clicks* y, por tanto, variar el tono (esto se hace fijando distintos valores para la opción BEEP).

La forma en que este programa controla la frecuencia de los sonidos es fijando el contenido de un contador (alrededor de 100 es el valor normalmente utilizado) y produ-

ciendo una salida por el altavoz cada vez que este contador se hace cero. Incluso colocando el contador a un valor de 100, éste tardará menos de 10 microsegundos, por lo que la frecuencia de salida del altavoz estará dentro del rango de frecuencias audibles (20 Hz a 20 KHz).

Si lo organizamos de forma que la variación del contador sea regular, aumentándolo o disminuyéndolo, obtendremos un sonido que cambia suavemente de tono.

El programa en código máquina que realiza esto es el siguiente:

**Variación del tono de salida de una forma suave**

```
F000           10     ORG   #F000
F000 F3        20     DI
F001 111000    30     LD     DE,#0010
F004 260A      40     LD     H,#0A
F006 3A485C    50     LD     A,(#5C48)

F009 1F        60     RRA
F00A 1F        70     RRA
F00B 1F        80     RRA
F00C 0EFE      90     LD     C,#FE
     ;
F00E EE10     100     XOR   #10
F010 D30C     110     OUT   (#C),A
F012 43       120     LD     B,E
F013 10FE     130     DJNZ  #F013
F015 25       140     DEC   H
F016 20F4     150     JR     NZ,#F00C
F018 1C       160     INC   E
F019 15       170     DEC   D
F01A 20E8     180     JR     NZ,#F004
F01C FB       190     EI
F01D C9       200     RET
```

Hay un par de apartados interesantes en el listado anterior. En primer lugar, el puerto de salida 254 fija el color del borde de la imagen y maneja el altavoz (véase *Manual del Spectrum*). Por tanto, para salvar el color del borde la línea F006h del programa hace el traslado de esta información desde la variable del sistema BORDCR, situada en la posición 5C48h al registro A; después colocamos los bits en las posiciones en que los necesitamos por las tres instrucciones siguientes:

La instrucción XOR de la posición F00Dh conmuta el bit del altavoz, tal y como se describe en el capítulo 6.

La instrucción DI situada al comienzo de la rutina y la EI colocada al final de la rutina se han puesto para evitar que la rutina sea interrumpida, mientras se ejecuta, por el teclado. Si esto ocurriese, las interrupciones modularían el tono de salida con un zumbido de 50 Hz, deteriorando la calidad del sonido producido.

Los registros H, DE y B se utilizan para controlar el tono de salida, la amplitud del espectro de variación de los tonos y la duración total. El registro E controla el tono; este es el registro fuente desde el cual el registro B es cargado rápidamente para poder utilizarlo en una operación DJNZ que controla el intervalo entre los *clicks* de salida del altavoz.

Decrementando el registro E, se producirá un tono ascendente; incrementándolo, se producirá un tono descendente.

El registro H controla el número de ciclos de una determinada señal de salida antes de efectuar el próximo incremento o decremento (variar el tono). Como resultado de esta función, el registro H también controla la duración total del programa.

El registro D controla el número de intervalos utilizados, es decir, la amplitud del espectro de variación del tono.

Todos los valores pueden ser variados de forma experimental y afectarán de una forma considerable al sonido producido, tal y como era de esperar.

Introduciendo de forma alternativa los valores 1Ch (28d) para realizar la instrucción "INC E" o 1Dh (29d) para realizar "DEC E", en la posición F018h (61464d), puedes obtener un sonido con un desplazamiento ascendente y descendente de tono que se puede parecer al producido por una sirena de policía.

Hay un segundo efecto sonoro que utiliza el cambio de frecuencia y que puede, por tanto, ser realizado con técnicas sencillas utilizando el comando BEEP. Esta rutina produce una salida de dos tonos diferentes al mismo tiempo. Al pensarla y desarrollarla, esperaba que esta salida produjese un sonido parecido al de una cuerda musical, pero el resultado no se parece a esto. Supongo que, para que parezca una cuerda musical, es necesario superponer dos formas de onda completas, en vez de superponer dos conjuntos de señales procedentes de conmutadores a diferentes frecuencias.

Sin embargo, este programa produce algunos efectos interesantes: desde una especie de gorgoritos graves, al sonido de una campana.

## Generación de notas dobles

```
F000              10     ORG    #F000
F000 F3           20     DI
F001 3A485C       30     LD      A,(#5C48)
F004 1F           40     RRA
F005 1F           50     RRA
F006 1F           60     RRA
F007 0600         70     LD      B,#00
F009 0EFE         80     LD      C,#FE
F00B 25           90     DEC    H          ;Contador bucle 1
F00C 2006        100     JR      NZ,#F014
F00E EE10        110     XOR    #10
F010 D30C        120     OUT    (#C),A
F012 26F0        130     LD      H,#F0    ;Fija contador
F014 2D          140     DEC    L          ;Contador bucle 2
F015 20F4        150     JR      NZ,#F00B
F017 EE10        160     XOR    #10
F019 D30C        170     OUT    (#C),A
F01B 2EED        180     LD      L,#ED    ;Fija contador
F01D 10EC        190     DJNZ   #F00B
F01F FB          200     EI
F020 C9          210     RET
```

El programa para generar el sonido con notas dobles utiliza el mismo sistema que vimos anteriormente, pero esta vez utilizamos dos contadores; uno para la nota que llamaremos 1 y el otro para la nota 2. La rutina descuenta cada contador de forma alterna-

tiva y, cada vez que uno de estos contadores se hace cero, aparecerá un *clik* en el altavoz, después de lo cual se volverá a inicializar el contador de nuevo.

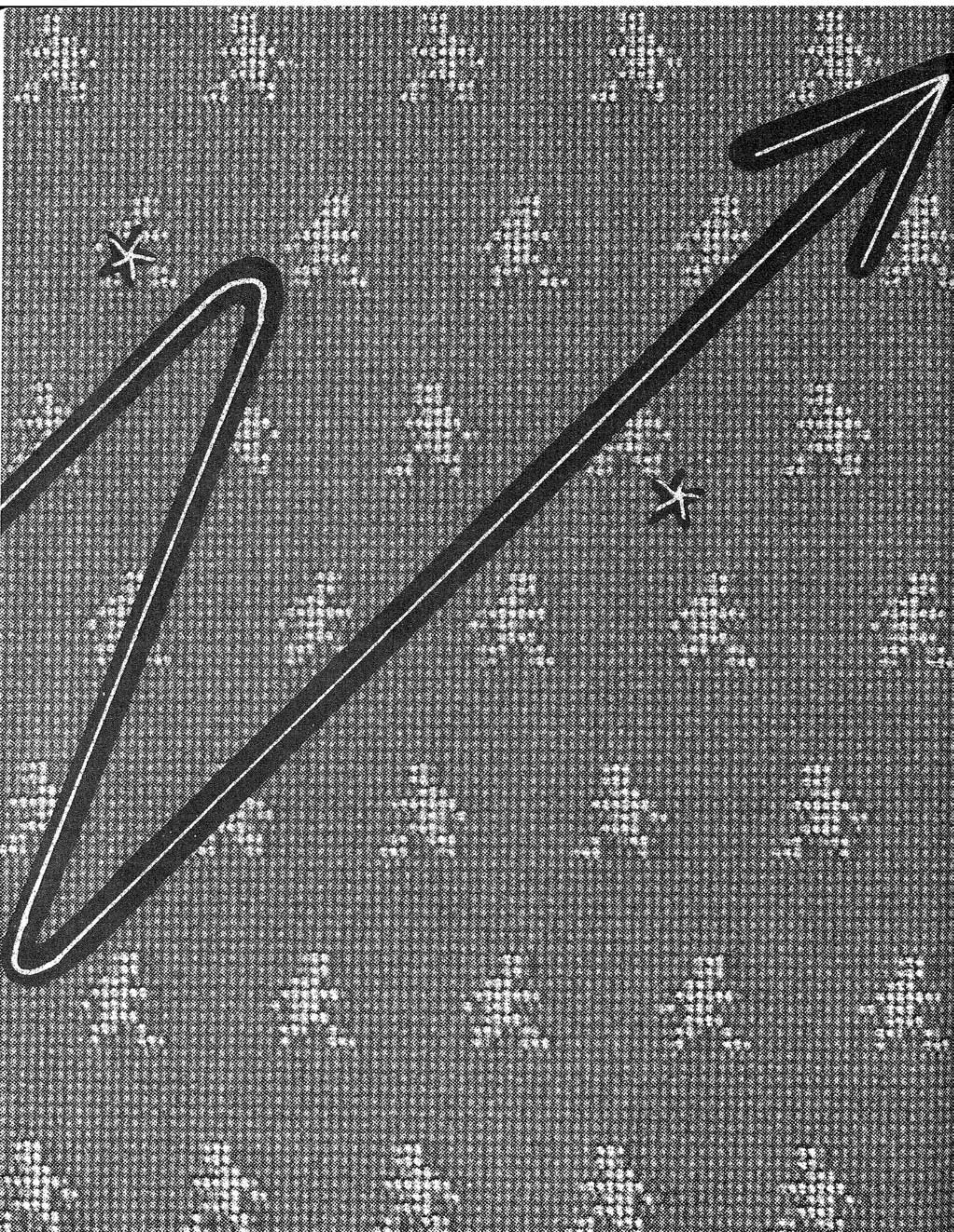
El número que se carga en la posición F006h del programa, en el registro B, controla el número de veces que se ejecuta el programa completo antes de pararse, es decir, controla la duración de la nota producida. Como sólo se utiliza el registro B, el mayor número con el que puedes trabajar será 256d, lo que equivale a decir que la duración es limitada. La duración de la nota depende también del tono producido, de forma que será mayor para una nota grave que para una aguda.

El programa en BASIC que viene a continuación trabaja con unas parejas particulares de notas. Estas son bastante variadas, pero las parejas más curiosas son aquellas en que ambas notas son casi iguales o cuando una de ellas es aproximadamente igual a un armónico de la otra.

Puedes utilizar todo esto como parte de un programa para tratamiento de interrupciones, pero el efecto queda un poco deteriorado por el hecho de que la nota aparecerá en ráfagas de cierta duración, para que el programa principal se pueda seguir ejecutando.

**Programa para la realización de notas dobles**

```
  100 FOR i=100 TO 250 STEP 50: F
OR j=1 TO 255
  110 POKE 61459,i: POKE 61468,j:
RANDOMIZE USR 61440
  120 PRINT AT 10,10;i;TAB 15;j
  130 NEXT j: CLS : NEXT i
```



# 14

# Estudio de un programa en código máquina. Conversión entre códigos hexadecimal y decimal

Hasta ahora hemos estado trabajando con rutinas que considerábamos que se utilizarían como subrutinas de grandes programas principales. Pero creo que puede resultar interesante trabajar con un programa completo en código máquina y analizar cómo es realizado y organizado para que sea accesible al usuario.

He intentado reconstruir en este capítulo la idea de crear un programa para realizar la conversión hexadecimal/decimal y viceversa. Yo he almacenado el programa en memoria utilizando mi ensamblador, de forma que me sea posible colocarlo en la zona de memoria que desee (tener un programa reubicable).

El programa utiliza el calculador del Spectrum para operar con dígitos hexadecimales o decimales. El tema de si es mejor realizar las operaciones aritméticas sencillas utilizando el calculador o escribiéndolas en el programa, da lugar a ideas dispares. Las operaciones más sencillas se pueden hacer utilizando las instrucciones "ADD", "SUB" o "SHIFT" para sumar, restar, multiplicar o dividir. Por ejemplo, para calcular el número que es "10 veces" uno dado, se puede hacer:

El número inicial está en el registro HL

```
ADD  HL,HL ; multiplica el número × 2.
PUSH HL    ; almacena el doble del número inicial en el stack.
ADD  HL,HL ; × 4.
ADD  HL,HL ; × 8.
POP  BC    ; recupera del stack el doble del número inicial.
ADD  HL,BC ; × 10.
```

Para realizar esto mismo utilizando las posibilidades de cálculo del Spectrum, se haría:

```
Número inicial en el registro BC
Código máquina.
CALL STACK_BC    CD 2B 2D  ; pasa el número al stack del calculador
RST 28           EF        ; utiliza el calculador.
                 A4        ; constante del stack "10"
                 04        ; multiplica
                 38        ; finaliza el cálculo
```

El número está ahora en la parte superior del *stack* del calculador, preparado para ser impreso utilizando la instrucción "CALL PRINT_FP", la cual realiza la impresión en números decimales, incluyendo la representación de números decimales o exponenciales, utilizando el símbolo "E", según sea conveniente en cada caso.

Las rutinas de manejo del calculador parecen un poco difíciles, ya que utilizan el Spectrum de la forma más directa (utilizando rutinas muy depuradas). Después de ejecutar la instrucción "RST 28", el programa no interpretará los bytes siguientes como códigos normales del Z80, sino como indicadores para llamar a unas determinadas rutinas de la ROM que realizan operaciones aritméticas u otro tipo de tareas.

El byte "A4" almacena el número 10 en el *stack* superior a los números anteriormente almacenados en dicho *stack*. "04" multiplica los dos números que fueron introducidos en la zona superior del *stack*, entre ellos, colocando el resultado en las posiciones que estos números ocupaban. El byte "38" indica el fin del cálculo y una vuelta a la forma normal de programación. Un buen generador de código máquina daría una lista completa de estos códigos o literales.

El programa de conversión "Hex/Dec" utiliza el *stack* del calculador para hallar los valores equivalentes de los números de entrada, en cada tipo de código.

Observando el programa, podrás darte cuenta de que tendrá que haber dos subrutinas principales, una para realizar la conversión hexadecimal/decimal y la otra para la conversión decimal/hexadecimal. Tendrá que haber también una rutina principal, que utilizará una u otra subrutina cuando corresponda; será la que tome la decisión sobre qué conversión ha de realizar.

Esta rutina principal no necesita ningún calculador; se trata únicamente de una rutina de selección con una única entrada. Si la entrada es "H" salta a una subrutina y si es "D" saltará a la otra.

Esta rutina podría ser la siguiente:

**Selección del tipo de conversión a realizar**

```
F000                 10        ORG   #F000
F000 FDCB01AE        20        RES   5,(IY+#01)
F004 FDCB016E        30        BIT   5,(IY+#01)
F008 28FA            40        JR     Z,#F004
F00A 3A085C          50        LD     A,(#5C08)
F00D FE48            60        CP    #48
F00F 2806            70        JR     Z,#F017
```

```
F011 FE44        80      CP     #44
F013 2853        90      JR     Z,#F068
F015 18E9       100      JR     #F000
```

En esta rutina volvemos a utilizar el mismo sistema para esperar a recoger el valor de la tecla pulsada que el que ya vimos en el capítulo 6, en la rutina para utilizar el Spectrum como una máquina de escribir. Cuando el código de la tecla está en el registro A, éste es comprobado dos veces; una para ver si es "H" (48h) y otra para ver si es "D" (44h). Cada uno de estos códigos provocaría un salto a una dirección determinada, dependiendo del código.

Sin embargo, esta rutina, tal y como está, no resulta comunicativa, ya que mostrará siempre una pantalla oscura. Por tanto, necesitaremos algún tipo de mensaje en la pantalla para poder responder con cualquier tipo de acción o saber en qué situación nos encontramos. Para realizar esto, añadimos las líneas de código siguientes, al principio de la anterior rutina.

**Selección del tipo de conversión a utilizar, con la impresión en pantalla de un mensaje**

```
F000             10      ORG    #F000
F000 3E02        20      LD     A,#02
F002 CD0116      30      CALL   #1601
                 40
F005 1100F1      50      LD     DE,#F100
F008 011E00      60      LD     BC,#001E
F00B CD3C20      70      CALL   #203C
F00E FDCB01AE    80      RES    5,(IY+#01)
F012 FDCB016E    90      BIT    5,(IY+#01)
F016 28FA       100      JR     Z,#F012
F018 3A085C     110      LD     A,(#5C08)
F01B FE48       120      CP     #48
F01D 2806       130      JR     Z,#F025
F01F FE44       140      CP     #44
F021 2853       150      JR     Z,#F076
F023 18E9       160      JR     #F00E
```

Las dos primeras instrucciones fijan la posición de impresión en la parte superior de la pantalla; hecho esto, se apunta a un mensaje con el registro DE, con la longitud que indica el registro BC, y se llama a la rutina PR_STRING (203Ch).

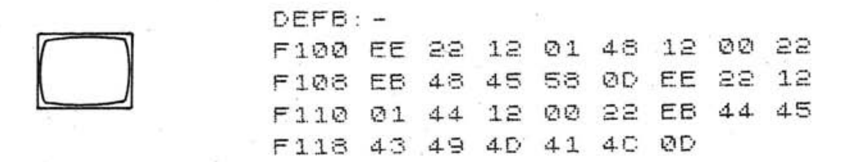
El código para el mensaje ha de introducirse en la dirección F100h. La longitud de este mensaje es de 30 bytes (1Eh) y todo el mensaje tomará el siguiente aspecto:

**Mensaje**

```
F100             10      ORG    #F100
F100 EE220148    20      DEFB   #EE,#22,#01,#48,#12,#00,#22
F107 EB484558    30      DEFB   #EB,#48,#45,#58,#0D,#EE,#22,#12
F10F 01441200    40      DEFB   #01,#44,#12,#00,#22,#EB,#44,#45
F117 43494D41    50      DEFB   #43,#49,#4D,#41,#4C,#0D
```

Realizando la impresión de los caracteres de este mensaje, que lo permiten, obtendremos:

**Mensaje impreso**



```
DEFB: -
F100 EE 22 12 01 48 12 00 22
F108 EB 48 45 58 0D EE 22 12
F110 01 44 12 00 22 EB 44 45
F118 43 49 4D 41 4C 0D
```

Una parte del texto resulta legible, pero una gran parte de éste no tiene sentido (parece que hubiera desaparecido). Los "bytes desaparecidos" contienen códigos de caracteres del Spectrum para realizar cosas que no son caracteres imprimibles. El byte "EEh" es el código de la palabra INPUT, tal y como puedes comprobar en el manual de *Sinclair*. "12h" es el código utilizado para "FLASH 1". Los dos bytes siguientes "12 00" son los correspondientes a "FLASH 0". El byte "EB" es el código de "FOR" completado con un espacio posterior.

El código completo se corresponderá con la línea en BASIC siguiente:

```
  10 PRINT "Teclea """; FLASH 1;
"H"; FLASH 0;""" para HEX."'"Tec
lea """; FLASH 1;"D"; FLASH 0;""
" para DECIMAL"
```

Esto aparecerá en la pantalla como:

INPUT "H" FOR HEX
INPUT "D" FOR DECIMAL

con las letras "H" y "D" parpadeando.

Puedes probar toda esta parte del programa colocando instrucciones RET en las posiciones F025h y F076h. Pero recuerda que la rutina está a la espera de las letras "D" o "H" en mayúsculas. Por tanto, tendremos que haber escrito algo para asegurarnos de que la función CAPS LOCK está activa (letras en mayúscula).

## Conversión hexadecimal/decimal

Ahora es el momento de entrar a considerar las dos subrutinas del programa. La conversión del código hexadecimal al decimal es probablemente la más sencilla; por tanto, la analizaremos en primer lugar.

Debido a que nosotros vamos a trabajar con la dirección más alta de la memoria, y ésta es la FFFFh, el programa nunca necesitará utilizar más de cuatro dígitos hexadecimales. Necesitamos organizar la entrada con un bucle de control que realiza la operación "× 4". En cada pasada del bucle, éste multiplicará el número total por 16d y después le añade el valor del dígito hexadecimal que se utilizó a la entrada. Lo organizaremos para que el valor total inicial sea 0 para que, después de cuatro pasadas, el valor total sea el de los cuatro dígitos hexadecimales y lo podamos visualizar en base decimal utilizando la rutina PRINT__FP.

El primer paso consiste en tomar el valor del dígito de entrada. Podemos utilizar de nuevo la rutina para "espera por una tecla" y colocar el código de la tecla pulsada en el registro A. Después nos aseguramos de que lo que hemos recogido es un dígito hexadecimal válido. Afortunadamente hay una pequeña subrutina en la ROM (en la posición 2D1Bh, que realiza la tarea de comprobar si la entrada es un número entre 0 y 9. Si el código de entrada no cumple esto, comprueba si se encuentra entre los valores A y F. El problema continuará solamente en el caso de que el código de entrada cumpla estos requisitos. Si la comprobación lo da como válido, imprimiremos el dígito de entrada).

### Entrada de un dígito en hexadecimal

```
E000              10      ORG   #E000
E000 FDCB01AE     20      RES   5,(IY+#01)
E004 FDCB016E     30      BIT   5,(IY+#01)
E008 28FA         40      JR     Z,#E004
E00A 3A085C       50      LD     A,(#5C08)
E00D CD1B2D       60      CALL  #2D1B
E010 3008         70      JR     NC,#E01A
E012 FE41         80      CP     #41       ;"A"
E014 38EA         90      JR     C,#E000
E016 FE47        100      CP     #47       ;"F"
E018 30E6        110      JR     NC,#E000
E01A F5          120      PUSH  AF
E01B D7          130      RST   #10        ;Imprime
E01C F1          140      POP   AF
E01D C9          150      RET
```

Ahora realizaremos la multiplicación y suma posterior. Necesitamos tener preparado el valor "16" en alguna parte del calculador para poder realizar la multiplicación. También necesitaremos tener un 0 en el *stack* del calculador al comenzar todas las operaciones. Todas estas cosas necesarias han de estar satisfechas antes de que comencemos a introducir dígitos.

El mejor lugar para almacenar el "16" es la zona de memoria para el calculador. Este número puede estar ahí todo el tiempo que queramos y podremos recuperarlo en el momento adecuado para introducirlo en el *stack* utilizando un sencillo literal. Lo introduces en la memoria metiéndolo en el *stack* y luego utilizando el literal "C0".

**Introducción del valor "16" en la memoria del calculador**

```
F032 3E10      200        LD   A,#10
F034 CD282D    210        CALL #2D28
F037 EF        220        RST  #28
F038 C0        230        DEFB #C0 ; Stack memoria 0
F039 38        240        DEFB #38 ; Fin calculo
F03A AF        250        XOR  A
F03B CD282D    260        CALL #2D28
```

Después de realizar la entrada de un dígito, tendremos en el registro A un dígito que será un número de 0 a 9 o una letra de la A a la F. Sin embargo, el valor del código de tecla "A" no es mayor en la unidad que el código de tecla "9", sino que es ocho veces mayor. Tenemos que hacer algunos ajustes más antes de que podamos estar seguros de que hemos tomado el valor correcto.

**Entrada en hexadecimal, sección primera**

```
F032 3E10      200        LD   A,#10
F034 CD282D    210        CALL #2D28
F037 EF        220        RST  #28
F038 C0        230        DEFB #C0 ; Stack memoria 0
F039 38        240        DEFB #38 ; Fin calculo
F03A AF        250        XOR  A
F03B CD282D    260        CALL #2D28
F041 FDCB01AE  300        RES  5,(IY+#01)
F045 FDCB016E  310 ESP1   BIT  5,(IY+#01)
F049 28FA      320        JR   Z,ESP1
F04B 3A085C    330        LD   A,(#5C08)
F04E CD1B2D    340        CALL #2D1B
F051 3008      350        JR   NC,#F05B
F053 FE41      360        CP   #41
F055 38EA      370        JR   C,#F041
F057 FE47      380        CP   #47
F059 30E6      390        JR   NC,#F041
F05B F5        400        PUSH AF
F05C D7        410        RST  #10
F05D F1        420        POP  AF
F05E FE41      430        CP   #41
F060 3802      440        JR   C,#F064
F062 D607      450        SUB  #07
```

Ahora introduciremos el valor del registro en el *stack*, ya que vamos a realizar operaciones aritméticas que dañarían el contenido de este registro y perderíamos su información. La primera operación que realizamos es la de multiplicar el valor almacenado en el *stack*, por 16.

EF RST 28
E0 introducir MEM,0 en el *stack* (este es el número 16).

04 multiplicar
38 fin de la operación

Ahora podemos sacar otra vez AF del *stack* y volverlo a introducir con la rutina STK_DIGIT.

```
F1         POP AF
CD 26 2D   CALL "STK_DIGIT" +
```

La última llamada (CALL) situada en la dirección 2D26h es una llamada modificada a la rutina STK_DIGIT, la cual almacena en el *stack* el valor de un carácter ASCII válido. Como el valor que hemos introducido puede que no sea un número (puede ser una letra de la A a la F), pero es un valor que nosotros sabemos que es válido, saltamos el procedimiento que hay en esta rutina para la comprobación de la validez del valor introducido (situado entre las posiciones 2D22h y 2D25h).

Ahora volvemos de nuevo al calculador.

```
EF   RST 28
OF   suma
38   fin del cálculo
```

Esto añade el valor inicial almacenado en el *stack* (ya multiplicado por 16) al nuevo valor que acabamos de introducir y mantiene el resultado en la parte más alta del *stack*.

Si lo preparamos todo para realizar esta operación cuatro veces, tendremos el valor de un número de cuatro dígitos hexadecimales en el *stack*, pudiendo imprimir este valor en decimal utilizando la rutina PRINT_FP.

Por tanto, como gran final, que sólo necesita una entrada para estar completa, la rutina resultante será:

**Rutina completa para la entrada en hexadecimal**

```
F032 3E10       200        LD    A,#10
F034 CD282D     210        CALL  #2D28
F037 EF         220        RST   #28
F038 C0         230        DEFB #C0 ; Stack memoria 0
F039 38         240        DEFB #38 ; Fin calculo
F03A AF         250        XOR   A
F03B CD282D     260        CALL  #2D28
                270
F03E 0604       280        LD    B,#04
F040 C5         290 ETI1   PUSH  BC
F041 FDCB01AE   300        RES   5,(IY+#01)
F045 FDCB016E   310 ESP1   BIT   5,(IY+#01)
F049 28FA       320        JR    Z,ESP1
F04B 3A085C     330        LD      A,(#5C08)
F04E CD1B2D     340        CALL   #2D1B
F051 3008       350        JR     NC,#F05B
```

```
F053 FE41        CP      #41
F055 38EA        JR      C,#F041
F057 FE47        CP      #47
F059 30E6        JR      NC,#F041
F05B F5          PUSH  AF
F05C D7          RST    #10
F05D F1          POP    AF
F05E FE41        CP      #41
F060 3802        JR     C,#F064
F062 D607        SUB    #07
F064 F5          PUSH  AF
F065 EF          RST    #28
F066 E0          DEFB #E0 ; Toma memoria 0
F067 04          DEFB #04 ; Multiplica
F068 38          DEFB #38 ; Fin calculo
F069 F1          POP    AF
F06A CD262D      CALL   #2D26
F06D EF          RST    #28
F06E 0F          DEFB #0F ; Suma
F06F 38          DEFB #38 ; Fin calculo
F070 C1          POP    BC
F071 10CD        DJNZ ETI1
F073 3E06        LD      A,06
F075 D7          RST    #10
F076 CDE32D      CALL   #2DE3
```

## Conversión decimal/hexadecimal

La segunda subrutina utilizada para realizar la conversión del decimal al hexadecimal sigue las mismas bases que la vista anteriormente, excepto que ahora se multiplica en cada pasada el total por 10, en vez de por 16. Tampoco tenemos preparada una rutina para imprimir los dígitos en hexadecimal, por lo que la tendremos que crear nosotros mismos.

Cada vez que extraemos un dígito hexadecimal del número con el que estamos trabajando, generaremos un valor en el registro A comprendido entre 0 y 15d (0 y Fh).

Añadiendo 48d (30h) tendremos los códigos para los números decimales. En el caso de los valores comprendidos entre 10 y 15, tenemos que organizarlo de manera que sumemos, además, 7 al número para conseguir los códigos de las letras de la "A" a la "F". El código de esta tarea quedará:

### Impresión de dígitos hexadecimales

```
E200             ORG    #E200
E200 FE0A        CP      #0A
E202 3802        JR      C,#E206
```

```
E204 C607          40          ADD   A,#07
E206 D7            50          RST   #10
E207 C9            60          RET
```

Necesitamos plantearnos también cómo resolver el problema de la extracción de los dígitos hexadecimales a partir del valor total del número decimal introducido. Esto resultará bastante sencillo de resolver. Suponte que nuestro valor está contenido en una pareja de registros (es necesario que sea una pareja de registros, ya que el máximo número con el que trabajaremos es FFFFh, el cual necesita dos registros para almacenarlo). El valor final que obtendremos será del tipo ''xxxx'', donde cada ''x'' simboliza un dígito hexadecimal. Necesitamos, por tanto, un programa sencillo para extraer cada dígito del número contenido en el par de registros de una forma ordenada, y enviarlo a la subrutina de impresión que hemos visto anteriormente.

Suponiendo que el valor está en el registro BC, la rutina siguiente efectuará todo el trabajo:

**Impresión de dígitos hexadecimales (1)**

```
E100              10          ORG   #E100
E100 78           20          LD     A,B
E101 E6F0         30          AND   #F0
E103 1F           40          RRA
E104 1F           50          RRA
E105 1F           60          RRA
E106 1F           70          RRA
E107 CD00E2       80          CALL  #E200   ;Rutina de impresion
E10A 78           90          LD     A,B
E10B E60F        100          AND   #0F
E10D CD00E2      110          CALL  #E200   ;Rutina de impresion
E110 79          120          LD     A,C
E111 E6F0        130          AND   #F0
E113 1F          140          RRA
E114 1F          150          RRA
E115 1F          160          RRA
E116 1F          170          RRA
E117 CD00E2      180          CALL  #E200   ;
E11A 79          190          LD     A,C
E11B E60F        200          AND   #0F
E11D CD00E2      210          CALL  #E200   ;
E120 C9          220          RET
```

La única desventaja de esta rutina es que no es reubicable, ya que formará parte de una subrutina que será llamada con instrucciones CALL y que estará situada en una determinada dirección. Podríamos superar este problema si lo organizamos todo utilizando un bucle de cuatro pasadas y vamos introduciendo los dígitos en el registro A, utilizando desplazamientos en vez de enmascararlos con operaciones lógicas AND.

## Impresión de dígitos hexadecimales (2)

```
E100            10       ORG   #E100
E100 1E04       20       LD     E,#04
E102 1604       30       LD     D,#04
E104 AF         40       XOR   A
E105 CB11       50       RL     C
E107 CB10       60       RL     B
E109 17         70       RLA
E10A 15         80       DEC   D
E10B 20F8       90       JR     NZ,#E105
E10D FE0A      100       CP     #0A
E10F 3802      110       JR     C,#E113
E111 C607      120       ADD   A,#07
E113 C630      130       ADD   A,#30
E115 D7        140       RST   #10
E116 1D        150       DEC   E
E117 20E9      160       JR     NZ,#E102
E119 C9        170       RET
```

Con los cambios hechos, nos resulta que esta segunda rutina es más corta y además es reubicable, aunque utiliza un registro más que la anterior. En conjunto, esta segunda parece la mejor, por lo que será la que usemos.

La primera parte de la rutina es una copia exacta de la que utilizamos para la conversión hexadecimal/decimal. Tú no tienes que colocar el número 10h (16d) en la memoria del calculador, ya que hay una constante 0Ah (10d) permanentemente preparada para ser llamada; esta constante forma parte de un grupo de constantes del sistema del Spectrum. Tampoco tendrás que volver a comprobar que los dígitos están comprendidos entre "A" y "F", ya que en este caso sólo pueden ser números ordinarios, al ser números decimales. Por tanto, la rutina quedará:

## Entrada en decimal (1.ª parte)

```
F084 AF          680      XOR   A
F085 CD282D      690      CALL  #2D28
                 700
F088 0605        710      LD     B,#05
F08A C5          720      PUSH  BC
F08B FDCB01AE    730      RES   5,(IY+#01)
F08F FDCB016E    740      BIT   5,(IY+#01)
F093 28FA        750      JR      Z,#F08F
F095 3A085C      760      LD     A,(#5C08)
F09F CD1B2D      810      CALL  #2D1B
F0A2 38E7        820      JR     C,#F08B
F0A4 F5          830      PUSH  AF
F0A5 D7          840      RST   #10
F0A6 F1          850      POP   AF
F0A7 F5          860      PUSH  AF
F0A8 EF          870      RST   #28
F0A9 A4          880      DEFB #A4 ; Constante 10
```

```
F0AA 04                DEFB #04 ; Multiplica
F0AB 38                DEFB #38 ; Fin calculo
F0AC F1                POP  AF
F0AD CD222D            CALL #2D22
F0B0 EF                RST  #28
F0B1 0F                DEFB #0F ; Suma
F0B2 38                DEFB #38 ; Fin calculo
F0B3 C1                POP  BC
F0B4 10D4              DJNZ #F08A
F0B9 CDA22D            CALL #2DA2
```

La última llamada (CALL) situada en la posición 2DA2h es para la rutina residente en la ROM, FP_TO_BC. Esta rutina coloca el valor del número en coma flotante situado en la parte superior del *stack* del calculador, en el registro BC. Desde este momento entra en juego un método sencillo para realizar la impresión del valor correspondiente en hexadecimal, tal y como ya hemos visto anteriormente. Antes de realizar la impresión realizamos, otra vez, un "TAB 16" para imprimir CHR 06h.

## Rutina completa para la entrada en decimal

```
F084 AF                XOR  A
F085 CD282D            CALL #2D28

F088 0605              LD   B,#05
F08A C5                PUSH BC
F08B FDCB01AE          RES  5,(IY+#01)
F08F FDCB016E          BIT  5,(IY+#01)
F093 28FA              JR   Z,#F08F
F095 3A085C            LD   A,(#5C08)
F098 FE0D              CP   #0D
F09A 2003              JR   NZ,#F09F
F09C C1                POP  BC
F09D 1817              JR   #F0B6
F09F CD1B2D            CALL #2D1B
F0A2 38E7              JR   C,#F08B
F0A4 F5                PUSH AF
F0A5 D7                RST  #10
F0A6 F1                POP  AF
F0A7 F5                PUSH AF
F0A8 EF                RST  #28
F0A9 A4                DEFB #A4 ; Constante 10
F0AA 04                DEFB #04 ; Multiplica
F0AB 38                DEFB #38 ; Fin calculo
F0AC F1                POP  AF
F0AD CD222D            CALL #2D22
F0B0 EF                RST  #28
F0B1 0F                DEFB #0F ; Suma
F0B2 38                DEFB #38 ; Fin calculo
F0B3 C1                POP  BC
F0B4 10D4              DJNZ #F08A
F0B6 3E06              LD   A,#06
F0B8 D7                RST  #10
F0B9 CDA22D            CALL #2DA2
F0BC 1E04              LD   E,#04
F0BE 1604              LD   D,#04
F0C0 AF                XOR  A
F0C1 CB11              RL   C
F0C3 CB10              RL   B
```

```
F0C5 17          1060      RLA
F0C6 15          1070      DEC   D
F0C7 20F8        1080      JR     NZ,#F0C1
F0C9 FE0A        1090      CP     #0A
F0CB 3802        1100      JR     C,#F0CF
F0CD C607        1110      ADD   A,#07
F0CF C630        1120      ADD   A,#30
F0D1 D7          1130      RST   #10
F0D2 1D          1140      DEC   E
F0D3 20E9        1150      JR     NZ,#F0BE
```

Esta última rutina ha ensamblado todos los componentes principales del programa completo. Aún tenemos que escribir las etiquetas para las dos subrutinas.

Observando, en conjunto, cómo están agrupadas las distintas partes del programa, resultará la siguiente organización:

| | |
|---|---|
| *Inicialización* | F000-F008 |
| Fija CAPS | |
| Abre canal pantalla | |
| *Mensaje* | F009-F911 |
| *Menú* | F012-F028 |
| Espera una tecla | |
| Elige "H" | |
| Elige "D" | |
| *Mensaje Hex./Dec.* | F029-F031 |
| *Composición calculador* | F032-F03D |
| *Valor Hex. entrada y cálculo* | F03E-F072 |
| *Fin rutina* | F073-F07A |
| Impresión "TAB 16" y número | |
| Salto a "finalizar" | |
| *Mensaje Dec./Hex.* | F07B-F083 |
| *Composición calculador* | F084-F087 |
| *Valor Dec. entrada y cálculo* | F088-F0B5 |
| *Cálculo e impresión en Hex.* | F0B6-F0D4 |
| *Finalización* | F0D5-F0E6 |
| Espera una tecla | |
| Resetea CAPS | |
| Return. | |

El listado completo quedará:

**Conversión Hexadecimal/Decimal**

```
F000             10        ORG   #F000
F000 FDCB30DE    20        SET   3,(IY+#30)
F004 3E02        30        LD     A,#02
F006 CD0116      40        CALL  #1601
```

```
F009 1100F1      50          LD    DE,MENS1
F00C 011300      60          LD    BC,#0013
F00F CD3C20      70          CALL  #203C
F012 FDCB01AE    80          RES   5,(IY+#01)
F016 FDCB016E    90          BIT   5,(IY+#01)
F01A 28FA       100          JR    Z,#F016
F01C 3A085C     110          LD    A,(#5C08)
F01F FE48       120          CP    #48
F021 2806       130          JR    Z,HEX
F023 FE44       140          CP    #44
F025 2854       150          JR    Z,DEC
F027 18E9       160          JR    #F012
F029 1113F1     170 HEX      LD    DE,MENS2
F02C 013A00     180          LD    BC,#003A
F02F CD3C20     190          CALL  #203C
F032 3E10       200          LD    A,#10
F034 CD282D     210          CALL  #2D28
F037 EF         220          RST   #28
F038 C0         230          DEFB #C0 ; Stack memoria 0
F039 38         240          DEFB #38 ; Fin calculo
F03A AF         250          XOR   A
F03B CD282D     260          CALL  #2D28
                270
F03E 0604       280          LD    B,#04
F040 C5         290 ETI1     PUSH  BC
F041 FDCB01AE   300          RES   5,(IY+#01)
F045 FDCB016E   310 ESP1     BIT   5,(IY+#01)
F049 28FA       320          JR    Z,ESP1
F04B 3A085C     330          LD    A,(#5C08)
F04E CD1B2D     340          CALL  #2D1B
F051 3008       350          JR    NC,#F05B
F053 FE41       360          CP    #41
F055 38EA       370          JR    C,#F041
F057 FE47       380          CP    #47
F059 30E6       390          JR    NC,#F041
F05B F5         400          PUSH  AF
F05C D7         410          RST   #10
F05D F1         420          POP   AF
F05E FE41       430          CP    #41
F060 3802       440          JR    C,#F064
F062 D607       450          SUB   #07
F064 F5         460          PUSH  AF
F065 EF         470          RST   #28
F066 E0         480          DEFB #E0 ; Toma memoria 0
F067 04         490          DEFB #04 ; Multiplica
F068 38         500          DEFB #38 ; Fin calculo
F069 F1         510          POP   AF
F06A CD262D     520          CALL  #2D26
F06D EF         530          RST   #28
                540
F06E 0F         550          DEFB #0F ; Suma
F06F 38         560          DEFB #38 ; Fin calculo
F070 C1         570          POP   BC
F071 10CD       580          DJNZ ETI1
                590
F073 3E06       600          LD    A,06
F075 D7         610          RST   #10
F076 CDE32D     620          CALL  #2DE3
F079 185A       630          JR    FIN
F07B 114DF1     640 DEC      LD    DE,MENS3
F07E 012800     650          LD    BC,#0028
F081 CD3C20     660          CALL  #203C
                670
F084 AF         680          XOR   A
```

```
F085 CD282D              CALL  #2D28
F088 0605                LD     B,#05
F08A C5                  PUSH  BC
F08B FDCB01AE            RES   5,(IY+#01)
F08F FDCB016E            BIT   5,(IY+#01)
F093 28FA                JR     Z,#F08F
F095 3A085C              LD     A,(#5C08)
F098 FE0D                CP     #0D
F09A 2003                JR     NZ,#F09F
F09C C1                  POP   BC
F09D 1817                JR     #F0B6
F09F CD1B2D              CALL  #2D1B
F0A2 38E7                JR     C,#F08B
F0A4 F5                  PUSH  AF
F0A5 D7                  RST   #10
F0A6 F1                  POP   AF
F0A7 F5                  PUSH  AF
F0A8 EF                  RST   #28
F0A9 A4                  DEFB #A4 ; Constante 10
F0AA 04                  DEFB #04 ; Multiplica
F0AB 38                  DEFB #38 ; Fin calculo
F0AC F1                  POP   AF
F0AD CD222D              CALL  #2D22
F0B0 EF                  RST   #28
F0B1 0F                  DEFB #0F ; Suma
F0B2 38                  DEFB #38 ; Fin calculo
F0B3 C1                  POP   BC
F0B4 10D4                DJNZ  #F08A
F0B6 3E06                LD     A,#06
F0B8 D7                  RST   #10
F0B9 CDA22D              CALL  #2DA2
F0BC 1E04                LD     E,#04
F0BE 1604                LD     D,#04
F0C0 AF                  XOR   A
F0C1 CB11                RL     C
F0C3 CB10                RL     B
F0C5 17                  RLA
F0C6 15                  DEC   D
F0C7 20F8                JR     NZ,#F0C1
F0C9 FE0A                CP     #0A
F0CB 3802                JR     C,#F0CF
F0CD C607                ADD   A,#07
F0CF C630                ADD   A,#30
F0D1 D7                  RST   #10
F0D2 1D                  DEC   E
F0D3 20E9                JR     NZ,#F0BE

F0D5 FDCB01AE      FIN   RES   5,(IY+#01)
F0D9 FDCB016E            BIT   5,(IY+#01)
F0DD 28FA                JR     Z,#F0D9
F0DF FDCB309E            RES   3,(IY+#30)
F0E3 3E0D                LD    A,13
F0E5 D7                  RST   #10
F0E6 C9                  RET
F100                     ORG   #F100
F100 434F4E56      MENS1 DEFM "CONVERSION? (H/D)"
F111 0D0D                DEFW #0D0D
F113 48455841      MENS2 DEFM "HEXADECIMAL"
F11E 0D0D                DEFW #0D0D
F120 5465636C            DEFM "Teclea 4 digitos."
F131 0D                  DEFB 13
F132 28436F6E            DEFM "(Con ceros a la izquierda)"
F14C 0D                  DEFB 13
```

```
F14D 44454349  1290 MENS3  DEFM "DECIMAL"
F154 0D0D      1300        DEFW #0D0D
F156 5465636C  1310        DEFM "Teclea hasta 5 digitos y ENTER"
F174 0D        1320        DEFB 13
```

El programa completo se puede utilizar haciendo una llamada (CALL) a la dirección F000h. Cuando sea utilizado desde un programa en BASIC, necesitarás utilizar "RANDOMIZE USR 61440".

# Apéndice A
# Rutinas en código máquina

## PaintCODE

Esta rutina, realizada en código máquina, coloreará cualquier línea, realizando así figuras en el Spectrum y permitiéndote producir figuras completamente coloreadas que de otra forma te resultarían difíciles de hacer.

Por desgracia, como todas las cosas agradables de la vida, esta rutina no es perfecta. Esto se debe a la forma en que esta rutina realiza su trabajo.

La rutina pinta la figura, observándola tal y como está en la pantalla, línea a línea y coloreando los puntos de imagen por parejas. Esto lo realiza poniendo a uno todos los bits situados entre parejas de puntos dibujados. Desgraciadamente, a veces se crea una cierta confusión al ejecutarse sobre una línea con un número impar de puntos dibujados, cuando no hay forma de indicar a la rutina qué puntos van unidos entre sí; esto da lugar a la aparición de líneas donde debería haber blancos, y viceversa.

Para eliminar este defecto se necesitaría un programa bastante más extenso que analizaría las líneas por cada lado, así como por el que ya ha sido cambiado. Pero supone un trabajo excesivo para las pocas veces en que se va a utilizar esta rutina.

### PaintCODE

```
F000                ORG   #F000
F000 210040         LD     HL,#4000
F003 1EC0           LD     E,#C0
F005 0E20           LD     C,#20
F007 0608           LD     B,#08
F009 7E             LD     A,(HL)
```

```
F00A 07          70      RLCA
F00B 3811        80      JR    C,#F01E
F00D 10FB        90      DJNZ  #F00A
F00F 77         100      LD    (HL),A
F010 23         110      INC   HL
F011 0D         120      DEC   C
F012 20F3       130      JR    NZ,#F007
F014 1D         140      DEC   E
F015 20EE       150      JR    NZ,#F005
F017 C9         160      RET
F018 0608       170      LD    B,#08
F01A 7E         180      LD    A,(HL)
F01B 07         190      RLCA
F01C 3008       200      JR    NC,#F026
                210
F01E 10FB       220      DJNZ  #F01B
F020 23         230      INC   HL
F021 0D         240      DEC   C
F022 28F0       250      JR    Z,#F014
F024 18F2       260      JR    #F018
F026 C5         270      PUSH  BC
F027 E5         280      PUSH  HL
F028 F5         290      PUSH  AF
F029 1806       300      JR    #F031
F02B 0608       310      LD    B,#08
F02D 7E         320      LD    A,(HL)
F02E 07         330      RLCA
F02F 3808       340      JR    C,#F039
F031 10FB       350      DJNZ  #F02E
F033 23         360      INC   HL
F034 0D         370      DEC   C
F035 20F4       380      JR    NZ,#F02B
F037 1816       390      JR    #F04F
F039 F1         400      POP   AF
F03A E1         410      POP   HL
F03B C1         420      POP   BC
F03C 1804       430      JR    #F042
F03E 0608       440      LD    B,#08
F040 7E         450      LD    A,(HL)
F041 07         460      RLCA
F042 CBC7       470      SET   0,A
F044 3811       480      JR    C,#F057
F046 10F9       490      DJNZ  #F041
F048 77         500      LD    (HL),A
F049 23         510      INC   HL
F04A 0D         520      DEC   C
F04B 20F1       530      JR    NZ,#F03E
F04D 1803       540      JR    #F052
F04F F1         550      POP   AF
F050 F1         560      POP   AF
F051 F1         570      POP   AF
F052 18C0       580      JR    #F014
F034 07         590      RLCA
F055 3804       600      JR    C,#F05B
F057 10FB       610      DJNZ  #F054
F059 18B4       620      JR    #F00F
F05B F5         630      PUSH  AF
F05C C5         640      PUSH  BC
F05D 1801       650      JR    #F060
F05F 07         660      RLCA
F060 10FD       670      DJNZ  #F05F
F062 77         680      LD    (HL),A
F063 C1         690      POP   BC
F064 F1         700      POP   AF
F065 18B7       710      JR    #F01E
```

## Trazador

Para dibujar los contornos que pueden luego ser rellenados con la rutina PaintCODE, deberías utilizar un programa como éste:

**Trazador**

```
  5 POKE 23660,5: LET a=0
 10 INPUT "x";x;"y";y
 20 PLOT  INVERSE a;x,y
 30 PRINT #1;AT 0,12;"3=</ 4=<\
9=>/ 0=>\"
 40 PAUSE 0
 50 LET y$=INKEY$
 60 LET x=x-(y$="3")-(y$="4")-(
y$="5")+(y$="8")+(y$="9")+(y$="0
")
 70 LET y=y-(y$="3")+(y$="4")-(
y$="6")+(y$="7")+(y$="9")-(y$="0
")
 80 IF y$="1" THEN  RANDOMIZE U
SR (5+PEEK 23635+256*PEEK 23636)
 90 LET x=x-(x>255)+(x<0): LET
y=y-(y>175)+(y<0)
100 GO TO 20
```

Es muy parecido al programa utilizado en el capítulo 8 para dibujar un *sprite*, pero éste es más compacto y te permitirá realizar dibujos en toda la pantalla.

El programa PaintCODE ha de colocarse en una declaración REM en la línea 1, tal y como se explicó en el capítulo 1. De nuevo se accede de forma indirecta, utilizando la variable del sistema PROG, de manera que tengas la dirección correcta, independientemente de que el Interfaz 1 esté conectado o no (véanse capítulos 1 y 6).

Puedes utilizar también el programa para borrar las líneas que han aparecido al utilizar la rutina PaintCODE debido a su mal funcionamiento.

## Encontrar Z$

En muchas ocasiones necesitas encontrar la dirección de una cadena de caracteres *(string)*, dentro de un programa en BASIC, durante la ejecución de un programa en código máquina. Hay rutinas para descubrir la dirección actual de la cadena de caracteres, pero han de ser acondicionadas para cada aplicación y en la ROM no hay una rutina que sea utilizable para esto.

Se pueden proponer muchas cosas para crear unas reglas generales, aplicables a la hora de copiar una cadena de caracteres en otra cadena que pueda ser direccionada por el código máquina. Una variable *string* adecuada para ser utilizada en esta aplicación es "Z$". Si pones la cadena de caracteres escogida o el elemento de un conjunto de cadenas en Z$, inmediatamente antes de llamar a la rutina en código máquina con "USR xxxxx", Z$ será siempre la última cadena en la zona de variables de la memoria, colocada delante de la variable del sistema E_LINE. Puedes entonces estar seguro de que encontrarás su dirección con una sencilla rutina de búsqueda.

Por desgracia, el código que indica el comienzo de una cadena de caracteres en las variables es el mismo que el de la versión en mayúsculas de la letra que denomina a la cadena (en este caso, la letra "Z"). Ya que existe la posibilidad de que esta letra pueda estar incluida en la cadena, el programa tiene que tener en cuenta esto para que no existan problemas. Esto se puede hacer controlando el carácter siguiente de cada carácter del que estábamos chequeando, el cual no será un carácter imprimible si la letra forma una variable *string* (excepto en el caso de que el *string* o cadena sea demasiado largo, ya que entonces, debido a la colocación del *string* en la memoria, pueden seguir produciéndose errores).

Si, por algún motivo, la búsqueda falla y el carácter "Z" no puede ser encontrado, el programa volverá al BASIC con un mensaje de error.

**Encontrar Z$**

```
F000                ORG   #F000
F000 3E5A           LD     A,#5A
F002 A7             AND   A
F003 2A595C         LD     HL,(#5C59)
F006 ED4B4B5C       LD     BC,(#5C4B)
F00A E5             PUSH  HL
F00B ED42           SBC   HL,BC
F00D 44             LD     B,H
F00E 4D             LD     C,L
F00F 0C             INC   C
F010 E1             POP   HL
F011 EDB9           CPDR
F013 2802           JR     Z,#F017
F015 CF             RST   #08
F016 01             DEFB  #01
F017 23             INC   HL
F018 23             INC   HL
F019 23             INC   HL
F01A 3E1F           LD     A,#1F
F01C BE             CP     (HL)
F01D 2B             DEC     HL
F01E 2B             DEC   HL
F01F 2B             DEC   HL
F020 3E5A           LD     A,#5A
F022 38ED           JR     C,#F011
F024 23             INC   HL
F025 23             INC   HL
F026 4E             LD     C,(HL)
F027 23             INC   HL
F028 46             LD     B,(HL)
F029 23             INC   HL
F02A C9             RET
```

La dirección del primer carácter de Z$ es devuelta en el registro HL y la longitud del *string* será devuelta en el registro BC.

## Rutina para generar señales acústicas de control del teclado

Una de las características más atractivas del Spectrum es la forma en la que responde cuando se pulsa una tecla, generando un débil sonido que lo asemeja a una máquina de escribir. Esto supone una gran ayuda para los usuarios que no han utilizado máquinas de escribir y que no pueden observar la pantalla cuando están utilizando el teclado. Con este sonido, el usuario está seguro de que el teclado ha aceptado la tecla pulsada.

La rutina que viene a continuación es una copia directa del método utilizado en la ROM para producir el citado sonido, de manera que puedes utilizar las mismas posibilidades cuando introduzcas rutinas en código máquina.

Todos los registros principales, así como el IX, son modificados por la rutina, por lo que es necesario realizar todas las instrucciones PUSH y POP que rodean a las tres instrucciones de la rutina.

Esta rutina puede colocarse como una subrutina en tus programas en código máquina y llamarla después de que se produzca una entrada desde el teclado.

### Rutina de generación de señales acústicas de control

```
F000               ORG   #F000
F000 DDE5          PUSH  IX
F002 E5            PUSH  HL
F003 D5            PUSH  DE
F004 C5            PUSH  BC
F005 110000        LD     DE,#0000
F008 21CB00        LD     HL,#00CB
F00B CDB503        CALL  #03B5
F00E C1            POP   BC
F00F D1            POP   DE
F010 E1            POP   HL
F011 DDE1          POP   IX
F013 C9            RET
```

Alternativamente, y si estás utilizando un programa en BASIC en el que puedes prescindir de una pareja de caracteres UDG, puedes intentar lo siguiente:

### Generación de señales acústicas de control en BASIC

```
  10 DATA 17,0,0,33,203,0,205,18
1,3,201
  20 FOR j=0 TO 9: READ n: POKE
USR "A"+j,n: NEXT j
  99 REM DEMOSTRACION
 100 PAUSE 0: RANDOMIZE USR USR
"A": GO TO 100
```

Esto te permite generar un *click* cuando está tomando un valor INKEY$, después de una "PAUSE 0", en lugar de la instrucción normal "INPUT...".

Como ya sabes, USR "A" únicamente define una dirección de la zona alta de la RAM correspondiente a un carácter definible por el usuario. Al utilizar la anterior forma de direccionamiento, no estás obligado a introducir los datos en ella utilizando una secuencia de ocho dígitos para realizar un carácter gráfico.

Como estamos trabajando con un programa en BASIC, no nos tenemos que preocupar de salvar y restaurar todos los registros.

# Apéndice B
# Algunas de las rutinas residentes en la ROM

**0010**
PRINT_A_1

Esta es la dirección llamada al ejecutar la instrucción "RST 10" (código de operación D7); esto supone dar control a la rutina principal de impresión. Esta debe ser la rutina más utilizada del Spectrum. Mostrará en la pantalla (en la posición indicada por el cursor) o enviará a la impresora (según el canal escogido) cualquier carácter que se pueda imprimir y que esté contenido en el registro A. Imprimirá las etiquetas expandidas correspondientes a los códigos de caracteres apropiados. Responderá a los 20 primeros caracteres de control del alfabeto de caracteres. De hecho, esta rutina realiza un trabajo muy amplio con la pantalla utilizada por el Spectrum.

**0028**
FP_CALC

Esta rutina es llamada al ejecutarse "RST 28" (código de operación EF). La instrucción "RST 28" accede al calculador, el cual es, en sí mismo, como un equipo aparte, con sus propias reglas. Esta rutina es otro punto de entrada a las rutinas de ampliación.

**0038**
MASK_INT

Es una rutina de interrupción enmascarable. Es activada, normalmente cada 20 microsegundos, para actualizar el reloj del Spectrum y chequear el teclado.

**03B5**
BEEPER

El registro HL contendrá el tono y el DE la duración. Es utilizada también para producir el sonido cuando se pulsa una tecla.

**04C2**
SA_BYTES

Esta es la rutina completa de almacenamiento en cassette (SAVE); es utilizada para almacenar todos los bytes. También es utilizada por la cabecera.

**0556**
LD_BYTES

Es la rutina de carga desde cassette (LOAD). El registro IX contendrá la dirección de comienzo y el registro DE contendrá el número de bytes que han de cargarse. El indicador de acarreo deberá estar a uno para hacer la carga; si está a cero, la rutina efectuará la operación VERIFY (verificación).

**0970**
SA_CONTRL

Es una rutina de almacenamiento. El registro IX contiene la dirección de comienzo y el registro DE el número de bytes a ser almacenados. Esta rutina contiene la rutina normal de almacenamiento completa, incluyendo la ''espera para una tecla''. Si se le llama para ser ejecutada a partir de la posición 0984h, se salta la espera hasta que se produzca una entrada.

**0D6B**
CLS

El canal ''S'' (número 2 para la zona superior de la pantalla) debe estar abierto antes y después de llamarla.

**0DD9**
CL_SET

Calcula y define DF_CC cuando los valores S_POSN están en el registro BC.

**0E44**
CL_LINE

Esta rutina limpiará un determinado número de líneas, comenzando desde la más inferior de la pantalla (línea 24). El número de líneas a limpiar debe estar especificado en el registro B. Con ''06 18'' se limpiará toda la pantalla.

**0EAC**
COPY

El canal de la impresora (el número 3) debe estar abierto antes de llamar a la rutina (véase rutina siguiente).

**1601**
CHAN_OPEN

Rutina para abrir un canal. Es utilizada de la forma siguiente: ''LD A,x    CALL 1601'': x = 1 para la parte inferior de la pantalla; x = 2 para la parte superior de la pantalla; x = 3 para la impresora.

**16C5**
SET_STK

Esta rutina limpia el *stack* del calculador. Es utilizada en operaciones del calculador.

**203C**
PR_STRING

Imprime cadenas de caracteres *(strings)* cuya dirección de comienzo está en el registro DE y la longitud estará en el registro BC. La rutina utiliza la instrucción ''RST 10'' para controlar los caracteres, etc.

**22AA**
PIXEL_AD

Coloca las coordenadas definidas por POINT en el registro BC y entrega la posición que ocupa el byte en el fichero de imagen, en el registro HL; también entrega en el registro A la posición que ocupa el punto de imagen dentro del byte.

**2D1B**
NUMERIC

Verifica que el registro A contiene un dígito ASCII, entre 0 y 9. El indicador de acarreo es puesto a 1 si este dígito es válido. Es utilizada en la rutina STK_NUM.

**2D22**
STK_DIGIT — Pasa un dígito válido en ASCII colocado en el registro A, al *stack* del calculador como un número en coma flotante.

**2D28**
**STACK_A** — Pasa el valor almacenado en el registro A al *stack* del calculador.

**2D2B**
STACK_BC — Pasa el valor almacenado en la pareja de registros BC al *stack* del calculador.

**2DA2**
FT_TO_BC — Pasa el valor en coma flotante situado en la parte alta del *stack,* a la pareja de registros BC.

**2DD5**
FP_TO_A — Es otra rutina para sacar valores del *stack*. Pasa el valor en coma flotante, situado en la zona alta del *stack,* al registro A.

**2DE3**
PRINT_FP — El valor en coma flotante, situado en la zona alta del *stack* del calculador, es impreso en decimal (incluyendo el punto decimal) o con la notación "E", según sea el caso.

Estas rutinas representan sólo una pequeña parte de las rutinas almacenadas en la zona ROM de la memoria y que son utilizadas en la realización, en el Spectrum, de los programas en BASIC.

Los libros que sirven de guía de la ROM del Spectrum te proporcionarán más información sobre este importante tema.

# Indice alfabético

S

**T**



**U**



**V**



**X**



**Z**